### 1、单例模式介绍

单例模式（Singleton），也叫单子模式。在应用这个模式时，单例对象的类必须保证只有一个实例存在。

许多时候整个系统只需要拥有一个全局对象，这样有利于我们协调系统整体的行为。比如在某个服务器程序中，该服务器的配置信息存放在一个文件中，这些配置数据由一个单例对象统一读取，然后服务进程中的其他对象再通过这个单例对象获取这些配置信息。这种方式简化了在复杂环境下的配置管理。

### 2、单例模式实现思路

* 一个类能返回对象一个引用(永远是同一个)和一个获得该实例的方法（必须是静态方法，通常使用getInstance这个名称）；
* 调用这个方法时，如果类持有的引用不为空就返回这个引用，如果类保持的引用为空就创建该类的实例并将实例的引用赋予该类保持的引用；
* 将该类的构造函数定义为私有方法，这样其他处的代码就无法通过调用该类的构造函数来实例化该类的对象，只有通过该类提供的静态方法来得到该类的唯一实例。

### 3、注意点

单例模式在多线程的，应用场合下必须小心使用。如果当唯一实例尚未创建时，有两个线程同时调用创建方法，那么它们同时没有检测到唯一实例的存在，从而同时各自创建了一个实例，这样就有两个实例被构造出来，从而违反了单例模式中实例唯一的原则。

解决这个问题的办法是为指示类是否已经实例化的变量提供一个互斥锁(虽然这样会降低效率)。

饱汉模式（懒加载）

饿汉模式（线程安全）

public class Singleton2 {

private static final Singleton2 *singleton* = new Singleton2();  
  
 private Singleton2() {  
 }  
  
 public static Singleton2 getInstance() {  
 return *singleton*;  
 }  
}

Holder模式

### 4、使用注意事项

1. 使用时不能用反射模式创建单例，否则会实例化一个新的对象
2. 使用懒单例模式时注意线程安全问题
3. 饿单例模式和懒单例模式构造方法都是私有的，因而是不能被继承的，有些单例模式可以被继承（如登记式模式）

### 5、适用场景

单例模式只允许创建一个对象，因此节省内存，加快对象访问速度，因此对象需要被公用的场合使用，如多个模块使用同一个数据源连接对象等等。如：

* 需要频繁实例化然后销毁的对象。
* 创建对象时耗时过多或者耗资源过多，但又经常用到的对象。
* 有状态的工具类对象。（频繁访问数据库或文件的对象）

以下都是单例模式的经典实用场景：

1. 资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如上述中的日志文件，应用配置。
2. 控制资源的情况下，方便资源之间的互相通信。如线程池等。