# 基本概念

这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。

注意：

1、单例类只能有一个实例。

2、单例类必须自己创建自己的唯一实例。

3、单例类必须给所有其他的对象提供这一实例。

# 意图

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

# 解决的问题

一个全局使用的类频繁地创建与销毁。

# 何时使用

当您想控制实例数目，节省系统资源的时候。

# 如何解决

判断系统是否已经有这个单例，如果有则返回，如果没有则创建。

# 优点

1、在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例。

2、避免对资源的多重占用（比如写文件操作）

# 缺点

没有接口，不能继承，与单一职责原则冲突，一个类应该只关心内部逻辑，而不关心外面怎么样来实例化。

# 使用场景

1、要求生产唯一的序列号。

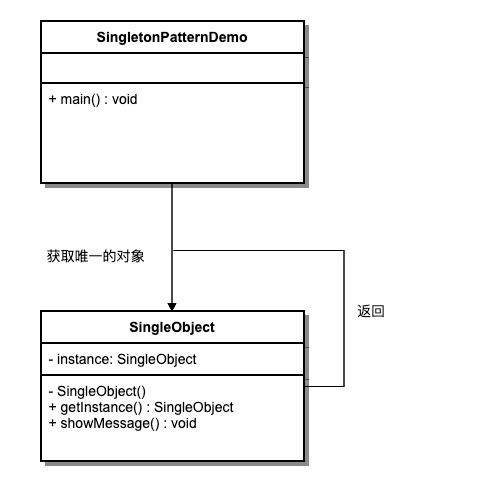
2、WEB中的计数器，不用每次刷新都在数据库里加一次，用单例先缓存起来。

3、创建的一个对象需要消耗的资源过多，比如 I/O 与数据库的连接等。

注意：getInstance() 方法中需要使用同步锁 synchronized (Singleton.class) ，防止多线程同时进入造成 instance 被多次实例化。

# 实现

我们将创建一个 SingleObject 类。SingleObject 类有它的私有构造函数和本身的一个静态实例。SingleObject 类提供了一个静态方法，供外界获取它的静态实例。SingletonPatternDemo 类使用 SingleObject 类来获取 SingleObject 对象。



1、创建一个 Singleton 类。



2、从 singleton 类获取唯一的对象。



3、执行程序，输出结果：



# 单例模式的实现方法

## 懒汉式，线程不安全

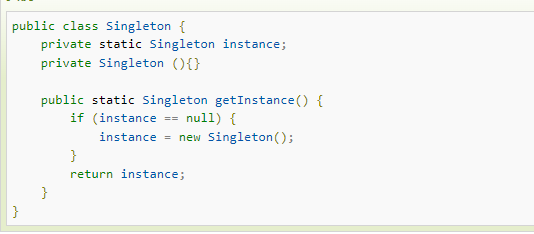
是否 Lazy 初始化：是

是否多线程安全：否

实现难度：易

描述：这种方式是最基本的实现方式，这种实现最大的问题就是不支持多线程。因为没有加锁 synchronized，所以严格意义上它并不算单例模式。这种方式 lazy loading 很明显，不要求线程安全，在多线程中不能正常工作。

### 代码示例



## 懒汉式，线程安全

是否 Lazy 初始化：是

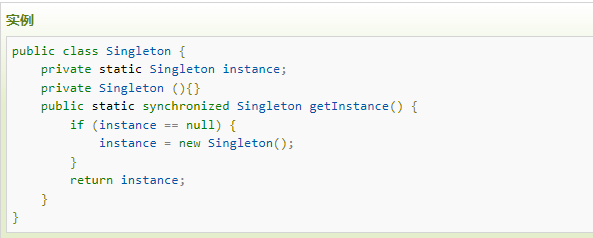
是否多线程安全：是

实现难度：易

描述：这种方式具备很好的 lazy loading，能够在多线程中很好的工作，但是效率很低，99% 情况下不需要同步。

优点：第一次调用才初始化，避免内存浪费。  
缺点：必须加锁 synchronized 才能保证单例，但加锁会影响效率。getInstance() 的性能对应用程序不是很关键（该方法使用不太频繁）。

### 代码示例



## 饿汉式

是否 Lazy 初始化：否

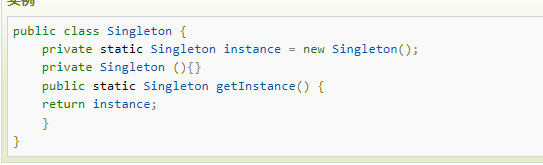
是否多线程安全：是

实现难度：易

描述：这种方式比较常用，但容易产生垃圾对象。  
优点：没有加锁，执行效率会提高。  
缺点：类加载时就初始化，浪费内存。

它基于 classloader 机制避免了多线程的同步问题，不过instance 在类装载时就实例化，虽然导致类装载的原因有很多种，在单例模式中大多数都是调用 getInstance 方法， 但是也不能确定有其他的方式（或者其他的静态方法）导致类装载，这时候初始化 instance 显然没有达到 lazy loading 的效果。

### 代码示例



## 双检锁/双重校验锁（DCL，即 double-checked locking）

JDK 版本：JDK1.5 起

是否 Lazy 初始化：是

是否多线程安全：是

实现难度：较复杂

描述：这种方式采用双锁机制，安全且在多线程情况下能保持高性能。getInstance() 的性能对应用程序很关键。

代码示例

