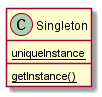
### 第五章：单例模式

全局变量和静态变量也可以实现只有一个实例化的对象。缺点在于这个对象必须提前创建，而不是实时创建的，很可能创建之后我们并不一定要用到它，也很有可能创建它很耗时或者很耗内存。

经典的单例模式编码如下：

|  |
| --- |
| public class Singleton{  private static Singleton uniqueInstance;  private Singleton(){}  public static Singleton getInstance(){  if(uniqueInstance == null)uniqueInstance = new Singleton();  return uniqueInstance;  }  } |

**单例模式确保一个类只有一个实例，并提供一个全局访问点。**



使用经典模式编码单例模式还需要注意防止多线程安全问题，产生两个实例，这种情况仅仅只在首次访问getInstance初始化时才会产生，因此加上synchronized不是一个好的选择，这会造成非首次使用之后的性能浪费。

1. 如果getInstance()的性能对应用程序不是很关键，可以使用synchronized。需要明确的是使用synchronized会使这个方法性能下降100倍，因此在频繁运行的地方需要慎重考虑。
2. 如果该单件实例经常创建，或者在创建和运行时方面的负担不太繁重，可以使用静态方式“急切创建”，如下：

|  |
| --- |
| public class Singleton{  private static Singleton uniqueInstance = new Singleton();  private Singleton(){}  public static Singleton getInstance(){return uniqueInstance;}  } |

1. 使用“双重检查加锁”，在getInstance()中减少使用同步

|  |
| --- |
| public class Singleton{  private volatile static Singleton uniqueInstance;  private Singleton(){}  public static Singleton getInstance(){  if(uniqueInstance == null){  synchronized(Singleton.class){  if(uniqueInstance == null){  uniqueInstance = new Singleton();  }  }  }  }  } |

volatile关键词确保：当uniqueInstance变量被初始化成Singleton实例时，多个线程正确地处理uniqueInstance变量。