# 单例模式（单例设计模式）详解

2020年4月1日　16:30

## 单例模式的定义与特点

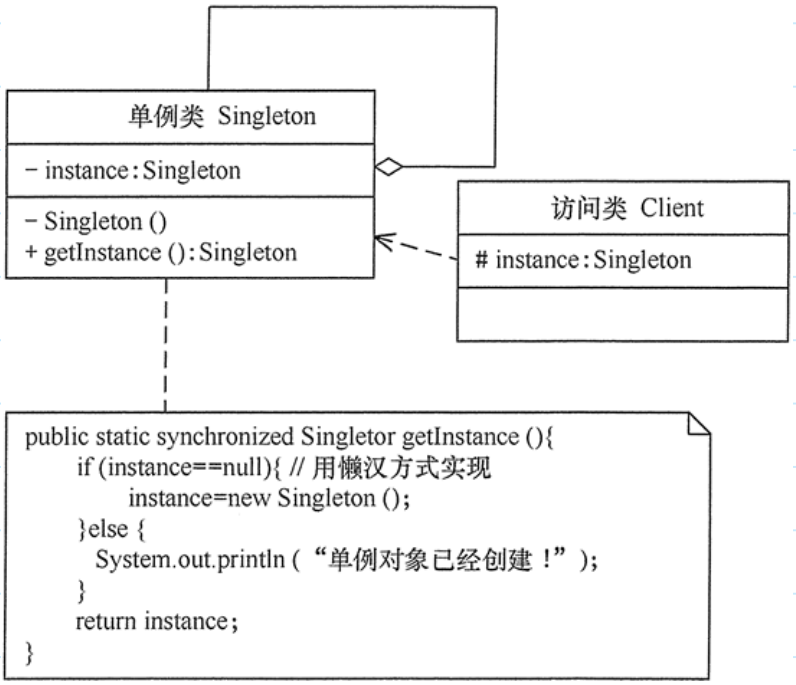
- **单例（Singleton）模式的定义：**
  指一个类只有一个实例，且该类能自行创建这个实例的一种模式。例如，Windows 中只能打开一个任务管理器，这样可以避免因打开多个任务管理器窗口而造成内存资源的浪费，或出现各个窗口显示内容的不一致等错误。
  在计算机系统中，还有 Windows 的回收站、操作系统中的文件系统、多线程中的线程池、显卡的驱动程序对象、打印机的后台处理服务、应用程序的日志对象、数据库的连接池、网站的计数器、Web 应用的配置对象、应用程序中的对话框、系统中的缓存等常常被设计成单例。

- **单例模式有 3 个特点：**
  - 单例类只有一个实例对象；
  - 该单例对象必须由单例类自行创建；
  - 单例类对外提供一个访问该单例的全局访问点；

## 单例模式的结构与实现

单例模式是设计模式中最简单的模式之一。通常，普通类的构造函数是公有的，外部类可以通过“new 构造函数()”来生成多个实例。但是，如果将类的构造函数设为私有的，外部类就无法调用该构造函数，也就无法生成多个实例。这时该类自身必须定义一个静态私有实例，并向外提供一个静态的公有函数用于创建或获取该静态私有实例。
下面来分析其基本结构和实现方法

- **单例模式的结构**
  单例模式的主要角色如下
  - 单例类：包含一个实例且能自行创建这个实例的类。
  - 访问类：使用单例的类。




- **单例模式的实现**
  - **懒汉式单例**

    该模式的特点是类加载时没有生成单例，只有当第一次调用 getInstance 方法时才去创建这个单例。

```
public class LazySingleton
{
    private static volatile LazySingleton instance=null;    //保证 instance 在所有线程中同步
    private LazySingleton(){}    //private 避免类在外部被实例化
    public static synchronized LazySingleton getInstance()
    {
        //getInstance 方法前加同步
        if(instance==null)
        {
            instance=new LazySingleton();
        }
        return instance;
    }
}
```

  - **饿汉式单例**

    该模式的特点是类一旦加载就创建一个单例，保证在调用 getInstance 方法之前单例已经存在了。饿汉式单例在类创建的同时就已经创建好一个静态的对象供系统使用，以后不再改变，所以是线程安全的，可以直接用于多线程而不会出现问题。

```
public class HungrySingleton
{
    private static final HungrySingleton instance=new HungrySingleton();
    private HungrySingleton(){}
```

```
    public static HungrySingleton getInstance()
    {
        return instance;
    }
}
```

- ## 单例模式的应用场景

前面分析了单例模式的结构与特点，以下是它通常适用的场景的特点。

- 在应用场景中，某类只要求生成一个对象的时候，如一个班中的班长、每个人的身份证号等。
- 当对象需要被共享的场合。由于单例模式只允许创建一个对象，共享该对象可以节省内存，并加快对象访问速度。如 Web 中的配置对象、数据库的连接池等。
- 当某类需要频繁实例化，而创建的对象又频繁被销毁的时候，如多线程的线程池、网络连接池等。

- ## 单例模式的扩展

单例模式可扩展为有限的多例（Multitcm）模式，这种模式可生成有限个实例并保存在 ArmyList 中，客户需要时可随机获取