单例模式

2020年2月24日 12:11

1.概念

1.1 定义

实现1个类只有1个实例化对象 & 提供一个全局访问点

1.2 作用

保证1个类只有1个对象，降低对象之间的耦合度

2. 模式原理

2.1 UML类图

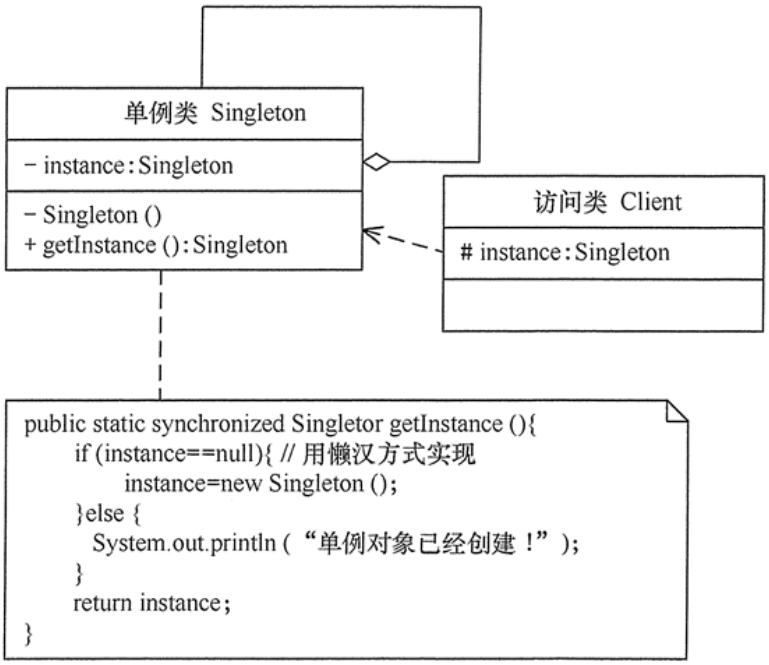

图2.1 单例模式UML类图

1. 单例类：包含一个实例且能自行创建这个实例的类。
2. 访问类：使用单例的类。

2.2 单例模式一般实现

```
public class Singleton {
//1. 创建私有变量 ourInstance (用以记录 Singleton 的唯一实例)
//2. 内部进行实例化
private static Singleton ourInstance = new Singleton();

//3. 把类的构造方法私有化，不让外部调用构造方法实例化
private Singleton() {}

//4. 定义公有方法提供该类的全局唯一访问点//5. 外部通过调用getInstance()方法来返回唯一的实例
public static Singleton newInstance() {
  return ourInstance;
}}
```

3. 实现方式

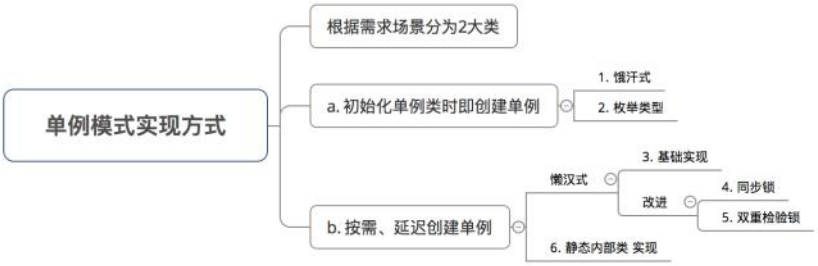

图3.1 单例模式实现方式

各种实现方式的示例参见设计模式课程。

4. 优/缺点

4.1 优点

1. 提供了对唯一实例的受控访问；
2. 由于在系统内存中只存在一个对象，因此可以节约系统资源，对于一些需要频繁创建和销毁的对象单例模式无疑可以提高系统的性能；
3. 可以根据实际情况需要，在单例模式的基础上扩展做出双例模式，多例模式；

4.2 缺点

1. 单例类的职责过重，里面的代码可能会过于复杂，在一定程度上违背了“单一职责原则”。
2. 如果实例化的对象长时间不被利用，会被系统认为是垃圾而被回收，这将导致对象状态的丢失。

5. 单例模式总结

应用场景	实现方式	原理	优点	缺点	备注
• 初始化时就需创建单例 • 单例对象 要求初始化速度快 & 占用内存小	1. 饿汉式	依赖JVM类加载机制，保证单例只被创建1次	• 线程安全 (即多线程下适用，因为 JVM 只会加载1次单例类) • 初始化速度快 • 占用内存小	单例创建时机不可控制	饿汉式 与 懒汉式最大区别 = 单例创建时机 • 饿汉式：单例创建时机不可控，即类加载时 自动创建 单例 • 懒汉式：单例创建时机可控，即有需要时，才 手动创建 单例
	2. 枚举类型	• 枚举类型 = 不可被继承的类（final） • 每个枚举元素 = 类静态常量 = 依赖JVM类加载机制，保证单例只被创建1次 • 枚举元素 都通过静态代码块来进行初始化 • 构造方法 访问权限 默认 = 私有（private） • 大部分方法都是final	• 线程安全 • 自由序列化 • 实现更加简单、简洁		
• 按需、延迟创建单例 • 单例初始化的操作耗时长 & 应用要求启动速度快 • 单例的占用内存比较大	3. 懒汉式 (基础实现)	1. 类加载时，先不自动创建单例 (即，将单例的引用先赋值为 Null) 2. 需要时才手动 创建 单例	• 按需加载单例 • 节约资源	线程不安全 (即多线程下不适用)	
	4. 同步锁 (懒汉式的改进)	使用同步锁 synchronized 锁住 创建单例的方法 (防止多个线程同时调用，从而避免造成单例被多次创建)	• 线程安全	造成过多的同步开销 (每次访问都要进行线程同步，加锁 = 耗时，耗能)	
	5. 双重校验锁 (懒汉式的改进)	• 校验锁1：若单例已创建，则直接返回已创建的单例，无需再执行加锁操作 • 校验锁2：防止多次创建单例问题	• 线程安全 • 节省资源（不需过多的同步开销）	实现复杂 (多种判断，易出错)	
	6. 静态内部类	• 按需加载：在静态内部类里创建单例，在装载该内部类时才会去创建单例 • 线程安全：类是由 JVM加载，而JVM只会加载1遍，保证只有1个单例	• 线程安全 • 节省资源（不需过多的同步开销） • 实现简单		

图5.1 单例模式总结图

6. 参考资料

[1] https://www.jianshu.com/p/b8c578b07fbc