单例模式（Singleton Pattern）日常工作中经常用到，先让我们一块看一下这几个问题

什么是单例模式呢，单例模式的主要目的是什么？

单例模式在多线程下会有什么问题？

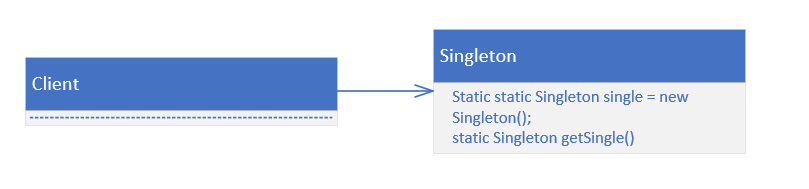
单例模式下的双重检验锁又是什么？

接下来让我们一块研究下单例模式，并寻找这些问题的答案吧

1、单例模式的定义

单例模式，保证一个类只有一个实例，并且自行实例化，并想整个系统提供这个实例。

通用的类图，如下：



2、使用单例模式有什么好处和问题

好处：单例模式在整个内存中只有一个实例化对象，对于频繁创建和销毁用户对象的场景，使用单例模式能够有很不错的性能提升

例如：Spring创建bean默认使用单例模式

缺点：单例模式可扩展性比较差，有变动只能修改代码

3、具体实现

单例模式的实现方式有：饿汉模式、简单懒汉模式（线程不安全）、线程安全的懒汉模式、双重检验锁的懒汉模式，让我们一块看下具体的实现吧

1、饿汉模式

优点:实现简单，类加载就实现了实例化，没有现成同步问题

缺点：类加载就实现了实例化，如果实例化对象一直不使用，会造成内存浪费

例子

public class Singleton {

public static final Singleton single = new Singleton ();

private Singleton (){}

public static Singleton getSingle()

{

return single;

}

public void doSomething()

{

System.out.println("i am single pattern");

}

}

2、简单懒汉模式（线程不安全）

优点：实现简单，第一次调用才会初始化，避免内存浪费

缺点：不支持多线程，高并发情况下会出现多个实例化对象

例子：

public class UnsafeSingleton {

public static UnsafeSingleton single ;

private UnsafeSingleton (){}

public static UnsafeSingleton getSingle()

{

if(single==null)

{

single = new UnsafeSingleton ();

}

return single;

}

public void doSomething()

{

System.out.println("i am unsafe single pattern");

}

}

3、懒汉模式，线程安全

优点：第一次调用才会初始化，避免内存浪费，支持多线程

缺点：需要加锁来保证单例的初始化，线程会进入阻塞状态，效率比较低

例子

public class SafeSingleton {

public static SafeSingleton single ;

private SafeSingleton (){}

public static synchronized SafeSingleton getSingle()

{

if(single==null)

{

single = new SafeSingleton ();

}

return single;

}

public void doSomething()

{

System.out.println("i am safe single pattern");

}

 }

4、懒汉模式，双重检验锁

优点：支持多线程，并且采用双检测机制，保持高性能

缺点：实现复杂

例子

public class SafeSingleton {

public static SafeSingleton single ;

private SafeSingleton (){}

public static SafeSingleton getSingle()

{

if(single==null)

{

synchronized(SafeSingleton.class)

{

if(single ==null)

{

single = new SafeSingleton ();

}

}

}

return single;

}

public void doSomething()

{

System.out.println("i am safe single pattern");

}

}

4、注意事项：

1、构造函数需要用private修饰，避免new创建

2、单例对象实例化过程，在多线程过程中，需要保证不会产生多个实例。

3、单例模式需要注意垃圾回收机制，一个单例对象在内存中长久不使用，jvm会认为这是一个垃圾对象，在CPU资源空闲情况下对对象进行清理。

4、如果在多线程情况下尽量使用双重检验锁的模式。