链滴

一 单例模式（海之魂研究学习系列——java设计模式）

作者：haihai

原文链接：https://ld246.com/article/1486475321338

来源网站：链滴

导航

什么是单例模式
单例模式的产生动机
单例模式使用场景举例
Java实现单例模式
单例模式的注意事项

一、什么是单例模式
单例模式最初的定义出现于《计模式》（艾迪生维斯理, 1994）：“保证一个仅有一个实例，并提供一个访问它的全局访问点。”
二、单例模式的产生动机
（1）资源共享的情况下，避免由于资源操作时导致性能或损耗等。如上述中的日志文件，应用配置。
（2）控制资源的情况下，方便资源之间的互相通信
（3）保证系统中一个类只有一个实例而且该实例易于外界访问，从而方便对实例个数的制并节约系统资源。如果希望在系统中某个类的对象只能存在一个，单例模式是最好的解决方案。
（4）其他等待网友共同发现哦，欢迎讨论，共享）
三、单例模式使用场景举例
（1）序列号生成器或资源管理器，比如操系统的文件系统，也是大的单例模式实现的具体例子，一个操作系统只能有一个文件系统。
（2）Windows操作系统的Task Manager（任务管理器）</pan>
（3）Windows操作系统的Recycle Bin（回收站）
（4）网站的计数器
（5）应用程序的日志应用，一都何用单例模式实现，这一般是由于共享的日志文件一直处于打开状态，因为只能有一个实例去操作否则内容不好追加
（6）Web应用的配置对象的读取，一般也应用单例模式，这个是由于配置文件是共享的资源。</pan>
（7）数据库连接池的设计一般是采用单例模式。数据库连接是一种数据库资源。数据库软件系统中使用数据库连接池，主要是节省开或者关闭数据库连接所引起的效率损耗，这种效率上的损耗还是非常昂贵的，因为何用单例模式来护，就可以大大降低这种损耗。
（8）多线程的线程池的设计一

也是采用单例模式，这是由于线程池要方便对池中的线程进行控制。
四、java实现单例模式

  一：单例类只能一个实例；二：单例类必须自行创建这个实例；三：单例类它必须自行向整个系统提供这个实例。
（1）懒汉式
<pre class="brush: java">public class SingletonClass{
    private static volatile SingletonClass intance=null;
    public static SingletonClass getInstance(){
        if(instance==null){
             synchonized(SingletonClass.class){
                   if(instance==null){
                   instance=new SingletonClass();
                    }
                  }
        }
            return instance;
    }
    private SingletonClass(){}
}

说明：</pre>

 一般用synchronized修饰第14行的getInstace()方法就可以了，干嘛要把synchronized块写到函数体里呢？原因如下：假设SingleTo这个类还有个synchronized的static函数f1()，那么，一旦调用f1()正在进行的时候，SingleTon的clas对象会被锁住，故而getInstance()方法就要等待了。同理，一旦调用getInstance()正在进行的时候，1()就无法被调用，只能等待getInstance()执行完了才能执行。所以把synchronized写到函数里面，际上是减小了锁的粒度。这样当instance已经被实例化了的时候，getInstance()这个函数是不会加锁，故不影响f1()函数的调用。

为啥要判断两次null == instance呢？把第15行的层if判断去掉不行吗？行！去掉也不会出错，但是如果判断两次性能更好。因为如果只判一次的话，每次执行getInstance()函数都要对synchronized块进行加锁；而如果进行两次判断的话只有第一次调用的时候（即instance==null）的时候会对synchronized块进行加锁，其他时候（如发现instance!=null）是不需要加锁的。

把第17行的内存if判断去掉不行吗？不！很有可能当外层判断的时候，instance确实是null；但是到了synchronized块中，执行第18行的instance = new SingleTon();的时候，instance已经被其他的线程给实例化了。

volatile是否有必要？
    如果只判断一次null==instance的话，就没要了。但如果像本博客代码里写的那样，判断了两次null==instance，那么volatile是必要的。
    为什么呢？举个反例，假设线程thread1走了第15行的if判断发现instance==null成立，于是都进入了外部的if体。这时候thread1先获取了synchronized块的锁，于是thread1线程会执行第18行的instance = new SingleTon();这句代码，问题就

在这里，这条语句它不是原子性执行的。在Java里，实例化一个对象的过程简单地讲，可以分为两步）先为instance对象分配一块内存，2）在这块内存里为instance对象里的成员变量赋值（比如第11里为url赋值）。假设当thread1执行完第1）步而还没有执行第2）步的时候，另外一个线程thread2到了第15行，这时候instance已经不是null了，于是thread2直接返回了这个instance对象。有什么题呢？instance对象的初始化（变量赋值等操作）还没执行完呢！thread2里直接得到了一个没有初化完全的对象，就有可能导致很严重的问题了。
    那么volatile关键字有啥作用呢？当用volatil修饰了instance变量之后，对instance的写操作”先行发生“于对它的读操作。（这是Ja a虚拟机里的先行发生原则，不明白的网友可以留言或是看看Java虚拟机的相关资料。）这样就保证，thread1中的instance变量被完全初始化之后，thread2才能读取它，当没有完成初始化时，thread只能等会儿啦。

 （2）饿汉
<pre class="brush: java">//对第一行static的一些解释
// java允许我们在一个类里面定义静态类。比如内部类（nested class）。
//把nested class封闭起来的类叫外部类。
//在java中，我们不能用static修饰顶级类（top level class）。
//只有内部类可以为static。
public class Singleton{
    //在自己内部定义自己的一个实例，只供内部调用
    private static final Singleton instance
nbsp;= new Singleton();
    private Singleton(){
        //do something
    }
    //这里提供了一个供外部访问本class的静态方法，可以直接访问
    public static Singleton getInstance(){
        return instance;
    }
}

</pre>
 (3) 其他实现方式：比如登记方式

五、单例模式的注意事项
（1）线程安全问题
（2）单例类不能被继承，扩展性问题
      使用注册式的单例，可以解决继承问题。/p>

六、总结
结合网络文章和自己的理解做了简单总结。后续会持续更新。欢迎大网友提出意见。

