软件体系结构

《软件体系结构作业六》

学　号 22920212204396

姓　名 黄子安

2024 年 4 月 16 日

1、用 Java 书写具有双向加锁功能的孤子模式（volatile ，synchronized）。

双重检查加锁可以在**多线程环境下**实现单例模式。为了满足多线程环境，如果直接在 getInstance 方法上加锁，会导致每次调用 getInstance 时都需要获得锁，这样会降低程序的性能。

而双重检查加锁允许只有在实例未被创建时才进行加锁操作，一旦实例被创建后，后续的调用就无需再加锁，从而提高程序的性能，具体的 getInstance 方法如下所述：

- 如果实例已经存在则直接获取实例，此时也肯定符合单例原则
- 如果实例尚未被创建，则在同步块内**再次检查实例是否为空**，避免可能有多个线程同时通过了第一次检查并创建了多个实例，从而保证只有一个线程获得了锁并创建实例，其他线程在获得锁之后便不再创建实例。

代码如下所示：

```
package singleton1;

public class Singleton {
    private static volatile Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

- 私有的静态变量 instance 和私有构造函数用来保存类的唯一实例。

- getInstance()方法是获取实例的全局访问点。其中进行双重检查加锁，只有获取到该对象的锁的线程才能执行代码块中的内容。其他线程如果想要执行同步代码块，也必须等待该对象的锁释放。
- 使用 volatile 关键字修饰 instance 变量，确保线程之间的可见性，即一个线程修改了 instance 的值，其他线程能够立即看到最新的值。

最后编写一个测试方法，在其中创建若干个线程，获取对应的实例

```
package singleton1;

public class Main {
    public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnable());
        Thread t2 = new Thread(new MyRunnable());
        Thread t3 = new Thread(new MyRunnable());
        t1.start();
        t2.start();
        t3.start();
    }

    static class MyRunnable implements Runnable {
        @Override
        public void run() {
            Singleton singleton = Singleton.getInstance();
            System.out.println("Thread: " + Thread.currentThread().getName()
                               + ", Singleton: " + singleton);
        }
    }
}
```

运行结果如下所示，可以发现不同的线程获得的实例地址相同，证明是同一个对象

```
D:\Java17\bin\java.exe "-javaagent:D:\IdeaU\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=59170:D:\IdeaU\IntelliJ IDEA
Thread: Thread-0, Singleton: singleton1.Singleton@3cd7d4f4
Thread: Thread-2, Singleton: singleton1.Singleton@3cd7d4f4
Thread: Thread-1, Singleton: singleton1.Singleton@3cd7d4f4

Process finished with exit code 0
```

2、用 Java 书写具有可变用例数目的孤子模式。

对本题的个人理解为是对单例模式的拓展，创建的对象实例可以不止一个但有数量上限，避免产生过多实例，如在某些场景下一个对象会占用大量的资源，此时并不想创建过多实例，但适当多个实例又可以保证效率比单例模式高。

具体实现如下，用一个数组列表存储对象，类提供一个对外的接口设置允许实例数量的最大值，之后通过双重检查加锁保证如果实例数量还未达到上限则继续创建，达到上限则直接返回最后一个实例

```
package singleton2;

import java.util.ArrayList;

public class Singleton {

    private static volatile ArrayList<Singleton> singletons = new ArrayList<>();
    public static volatile int num = 0;

    public static void setNum(int num) {
        Singleton.num = num;
    }

    public static Singleton getInstance() {
        if (singletons.size() < num) {
            synchronized (Singleton.class) {
                if (singletons.size() < num) {
                    singletons.add(new Singleton());
                }
            }
        }
        return singletons.get(singletons.size() - 1);
    }
}
```

测试方法如下：

```
package singleton2;

public class Main {
    public static void main(String[] args) {
        Singleton.setNum(5);
        for (int i = 0; i < 10; i++) {
            new Thread(() -> System.out.println("Thread: " +Thread.currentThread().getName()
                    + ", Singleton: " + Singleton.getInstance())).start();
        }
    }
}
```

运行结果如下图所示，可以保证最多只会出现 5 个不同的实例

```
D:\Java17\bin\java.exe "-javaagent:D:\IdeaU\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=62349:D:\IdeaU\IntelliJ
Thread: Thread-1, Singleton: singleton2.Singleton@3227ba97
Thread: Thread-4, Singleton: singleton2.Singleton@6f8dff11
Thread: Thread-2, Singleton: singleton2.Singleton@42d2060
Thread: Thread-6, Singleton: singleton2.Singleton@6f8dff11
Thread: Thread-8, Singleton: singleton2.Singleton@6f8dff11
Thread: Thread-5, Singleton: singleton2.Singleton@6f8dff11
Thread: Thread-0, Singleton: singleton2.Singleton@2461b908
Thread: Thread-3, Singleton: singleton2.Singleton@dcb54
Thread: Thread-9, Singleton: singleton2.Singleton@6f8dff11
Thread: Thread-7, Singleton: singleton2.Singleton@6f8dff11
```