JAVA 编程进阶上机报告

第二次上机作业

学　　院智能与计算学部
专　　业　软件工程
姓　　名　3018216056
学　　号　3018216056
年　　级　2018 级
班　　级　软件工程一班

首先考虑到只能该类只能有一个实例，构造方法必须**私有**，不能暴露给外界。因此想到可以维护一个**私有静态**的对象，向外界提供获得该对象的方法，代码如下：

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

    // some attributes and methods

    private Singleton() {
        // do something
    }

    public static synchronized Singleton getInstance() {
        return instance;
    }
}
```

但这种方法要求构建类时就创造一个对象，如果该对象需要的资源较多，可能对系统资源造成浪费，故可以在其第一次使用时再创建，代码如下：

```
public class Singleton {
    private static Singleton instance = null;

    // some attributes and methods

    private Singleton() {
        // do something
    }

    public static Singleton getInstance() {
        if (instance == null) instance = new Singleton();
        return (instance == null ? new Singleton(): instance);
    }
}
```

进一步分析发现，这种实现方式在多线程情况下可能实例化多个对象，对其加同步锁保护：

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

但此时每次调用方法都需要加锁，对效率有影响，实际上只有实例化的部分需要加锁，改进后：

```
    public static  Singleton getInstance() {
        if (instance == null)
        {
            synchronized(Singleton.class)
            {
                // judge again if instance is instantiated
                if (instance == null)
                instance = new Singleton();
            }
        }
        return instance;
    }
```

这样，只有在 `instance` 未被实例化之前调用该方法才会加锁，一旦对象创建成功，再调用该方法会返回 `instance` 对象，不会再进入 `synchronized` 的代码段，提高了效率。

查阅相关资料得知了解了**懒汉式**、**饿汉式**等相关概念，各有其试用场景，根据具体问题分析。另外还了解到除了线程安全的问题外，单例还可能被反射和序列化等破坏，在构造函数中补充对反射攻击的应对：

```
    private Singleton() {
        if (instance != null) {
            throw new RuntimeException("instantiate more than one time");
        }
        // do something
    }
```

两种主要的实现方式总结如下：

饿汉式：

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

    // some attributes and methods

    private Singleton() {
        if (instance != null) {
            throw new RuntimeException("instantiate more than one time");
        }
        // do something
    }

    public static synchronized Singleton getInstance() {
        return instance;
    }
}
```

懒汉式：

```
public class Singleton {
    private static Singleton instance = null;

    // some attributes and methods

    private Singleton() {
        if (instance != null) {
            throw new RuntimeException("instantiate more than one time");
        }
        // do something
    }

    public static  Singleton getInstance() {
        if (instance == null)
        {
            synchronized(Singleton.class)
            {
                // judge again if instance is instantiated
                if (instance == null)
                instance = new Singleton();
            }
        }
        return instance;
    }
}
```

通过本次实验，了解体会了单例设计模式，思考学习了其实现方式，并查阅相关资料了解了其中可能存在的一些问题，提升了对实际工程项目的理解。