JAVA设计模式之单例模式

一、单例模式的介绍

Singleton是一种创建型模式，指某个类采用Singleton模式，则在这个类被创建后，只可能产生一个实例供外部访问，并且提供一个全局的访问点。

全局对象和Singleton模式有本质的区别，因为大量使用全局对象会使得程序质量降低，而且有些编程语言根本不支持全局变量。最重要的是传统的全局对象并不能阻止一个类被实例化多次。

二、单例模式的特点

- 单例类只能有一个实例
- 单例类必须自己创建自己的唯一实例。
- 单例类必须给所有其他对象提供这一实例。

三、单例模式的应用

- 每台计算机可以由若干个打印机，但只能有一个Printer Spooler，避免有两个作业同时输出到打印机。
- 一个具有自动编号主键的表可以有多个用户同时使用，但数据库中只能有一个地方分配下一个主键。否则会出现主键重复。

四、单例模式使用的注意

- 不要使用单例模式存取全局变量。这违背了单列模式的用意，最好放到对应类的静态成员中。
- 不要将数据库连接做成单例，因为一个系统可能与数据库有多个连接，并且在有连接池的情况下，应当尽可能及时释放连接。Singleton模式由于使用静态成员存储类实例，所以可能会造成资源无法及时释放。

五、单例模式的举例

（1）

```
/* 线程安全 但效率比较低 一开始就要加载类new一个 对象
  这是饿汉方式的单例模式*/

public class Singleton1 {
    private Singleton1(){
    }
    private static final Singleton1 instance=new Singleton1();
    public static Singleton1 getInstancei(){
        return instance;
    }
}
```

（2）

```
// 饱汉方式的单例模式 但是有多个线程访问时就不是安全的 返回的不是同一个对象
public class Singleton {
    private Singleton(){
    }
    private static Singleton instance;
    public static Singleton getInstance() {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
}
```

（3）

```
//虽然是安全的 但是效率非常低在一个时候只有一个线程能访问 同时返回一个对象
public class Singleton2 {
    private Singleton2(){
    }
    private static Singleton2 instance;
    public static synchronized Singleton2 getInstance() {
        if (instance == null)
            instance = new Singleton2();
        return instance;
    }
}
```

（4）

```
/* 线程安全 并且效率高 能有多个线程访问*/
public class Singleton3 {
    private static Singleton3 instance;
    private Singleton3() {
    }
    public static Singleton3 getIstance() {
        if (instance == null) {
            synchronized (Singleton3.class) {
                if (instance == null) {
                    instance = new Singleton3();
                }
            }
        }
        return instance;
    }
}
```