单件模式(Singleton Pattern)

已经学习到的***设计原则：***

- 抽象且封装变化部分，使之与不变化部分分离开来。
- 面对接口编程，而不是面对实现编程
- 多用组合，少用继承
- 对扩展开放，对修改关闭(开闭原则)
- 依赖倒置原则(Dependency Inversion Principle)
- 为交互对象之间的松耦合儿努力
- 依赖抽象，不要依赖具体类

已经学习到的***设计模式：***

- 创建型模式
    - 简单工厂模式
    - 工厂方法模式
    - 抽象工厂模式
- 行为型模式
    - 策略模式
    - 观察者模式
- 结构型模式
    - 装饰者模式

今天继续学习一种创建型模式——单件模式。

定义：确保一个类只有一个实例，且提供一个全局访问点。

单件模式的类很简单，直接看代码：

```
1. public class Singleton {
2.     private static Singleton instance = null;
3.     private Singleton() {} //单件模式的构造器是私有的
4.     public static Singleton getInstance() {
5.         if (instance == null) {
6.             instance = new Singleton();
7.         }
8.         return instance;
9.     }
10. }
```

从上面的代码来看，单件模式满足三个条件：

- 类构造器私有
- 存在一个静态的私有变量作为单例对象
- 存在一个静态方法用来实例化并且获取单例对象

不足：
显然上面的单例模式不是线程安全的，假设线程A第一次访问getInstance()方法，且判定instance为null，这个时候线程B闯入，此时线程A还未创建对象成功，线程B也会进入创建对象代码块。此时两个线程就创建来两个对象，显然这是违背设计初衷的。
存在三个解决方法：

- 饿汉式初始化：即在类加载初期就完成单例对象的创建。看代码：

```
1. public class Singleton {
2.     private static Singleton instance = new Singleton();
3.     private Singleton() {} //单件模式的构造器是私有的
4.     public static Singleton getInstance() {
5.         return instance;
6.     }
7. }
```

这种方式比较高效，但是单例模式设计初衷就是推迟单例对象的创建，虽然有点违背，但保证对象的唯一性；

- 同步getInstance()方法：一种简单的想法就是同步getInstance()方法，同步这种方法之后，保证了只有一个线程能够访问这个方法。看代码：

```
1. public class Singleton {
2.     private static Singleton instance = null;
3.     private Singleton() {} //单件模式的构造器是私有的
4.     public synchronized static Singleton getInstance() {
5.         if (instance == null) {
6.             instance = new Singleton();
7.         }
8.         return instance;
9.     }
10. }
```

这种方法的优点就在于保证了在多线程访问的情况下也能保证只有一个对象能够创建。但是缺点也很明显，就是低效性，尤其是在每次需要获得对象是都会进入同步方法，这会大大降低程序运行效率。

- 同步创建对象代码块(双重检查): 这种方法避免了上述方法的缺点，只有在创建对象的时候上锁而不是对方法上锁。看代码：

```
1. public class Singleton {
2.     private volatile static Singleton instance = null;
3.     private Singleton() {} //单件模式的构造器是私有的
4.     public static Singleton getInstance() {
5.         if (instance == null) {
6.             synchronied(Singleton.class) {
7.                 if (instance == null) {//double check
8.                     instance = new Singleton();
9.                 }
10.             }
11.         }
12.         return instance;
13.     }
14. }
```

这种方法保证了只有在创建对象对时候上锁，而之后的每次获取对象都不会进入同代码块。但值得注意的是必须讲单件对象的声明为volatile，这是必须的，代表对象创建过程对其他线程可见。