设计模式GOF23

- **将设计者的思维融入大家的学习和工作中，更高层次的思考！**

- **创建型模式：**
  - 单例模式、工厂模式、抽象工厂模式、建造者模式、原型模式。
- **结构型模式：**
  - 适配器模式、桥接模式、装饰模式、组合模式、外观模式、享元模式、代理模式。
- **行为型模式：**
  - 模版方法模式、命令模式、迭代器模式、观察者模式、中介者模式、备忘录模式、解释器模式、状态模式、策略模式、职责链模式、访问者模式。

单例模式

- 核心作用：
  - 保证一个类只有一个实例，并且提供一个访问该实例的全局访问点。

- 常见应用场景：
  - Windows的Task Manager（任务管理器）就是很典型的单例模式
  - windows的Recycle Bin（回收站）也是典型的单例应用。在整个系统运行过程中，回收站一直维护着仅有的一个实例。
  - 项目中，读取配置文件的类，一般也只有一个对象。没有必要每次使用配置文件数据，每次new一个对象去读取。
  - 网站的计数器，一般也是采用单例模式实现，否则难以同步。
  - 应用程序的日志应用，一般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。
  - 数据库连接池的设计一般也是采用单例模式，因为数据库连接是一种数据库资源。
  - 操作系统的文件系统，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。
  - Application 也是单例的典型应用（Servlet编程中会涉及到）
  - 在Spring中，每个Bean默认就是单例的，这样做的优点是Spring容器可以管理
  - 在servlet编程中，每个Servlet也是单例
  - 在spring MVC框架/struts1框架中，控制器对象也是单例

单例模式

- 单例模式的优点：
  - 由于单例模式只生成一个实例，减少了系统性能开销，当一个对象的产生需要比较多的资源时，如读取配置、产生其他依赖对象时，则可以通过在应用启动时直接产生一个单例对象，然后永久驻留内存的方式来解决
  - 单例模式可以在系统设置全局的访问点，优化环共享资源访问，例如可以设计一个单例类，负责所有数据表的映射处理

- 常见的五种单例模式实现方式：
  - 主要：
    - 饿汉式（线程安全，调用效率高。 但是，不能延时加载。 ）
    - 懒汉式（线程安全，调用效率不高。 但是，可以延时加载。 ）
  - 其他：
    - 双重检测锁式（由于JVM底层内部模型原因，偶尔会出问题。不建议使用）
    - 静态内部类式(线程安全，调用效率高。 但是，可以延时加载)
    - 枚举单例(线程安全，调用效率高，不能延时加载)

单例模式

- 饿汉式实现（单例对象立即加载）

```
public class SingletonDemo02 {
        private static  /*final*/   SingletonDemo02 s = new SingletonDemo02();

        private SingletonDemo02(){} //私有化构造器

        public static /*synchronized*/ SingletonDemo02 getInstance(){
                return s;
        }
}
```

```
public class Client {
        public static void main(String[] args) {
                SingletonDemo02 s = SingletonDemo02.getInstance();
                SingletonDemo02 s2 = SingletonDemo02.getInstance();
                System.out.println(s==s2); //结果为true
        }
}
```

- 饿汉式单例模式代码中，static变量会在类装载时初始化，此时也不会涉及多个线程对象访问该对象的问题。虚拟机保证只会装载一次该类，肯定不会发生并发访问的问题。因此，可以省略synchronized关键字。

- 问题：如果只是加载本类，而不是要调用getInstance()，甚至永远没有调用，则会造成资源浪费！