# 单例模式

1. 介绍

单例模式(Singleton)是一种对象创建模式，用于产生一个对象的实例，可以确保系统中一个类只产生一个实例，在java中单例的范围是一个虚拟机的范围。单例模式是设计模式中最简单的一种，但是应用广泛，对于初学设计模式的人来说，有很多陷阱。

美国宪法规定了美国总统的选举方式，任期和继承的顺序，但是在任何时候，最多只有一个总统，而总统本身又是对外访问的一个“点”。

1. 实现方式

实现单例模式的思路是：   
    一个类能返回对象一个引用(永远是同一个)和一个获得该实例的方法（必须是静态方法，通常使用getInstance这个名 称）；当我们调用这个方法时，如果类持有的引用不为空就返回这个引用，如果类保持的引用为空就创建该类的实例并将实例的引用赋予该类保持的引用；同时我们 还将该类的构造函数定义为私有方法，这样其他处的代码就无法通过调用该类的构造函数来实例化该类的对象，只有通过该类提供的静态方法来得到该类的唯一实例。

1. 定义一个私有private实例变量；
2. 定义一个private或ptotected构造方法；
3. 定义一个外部可获得类的实例的static方法
   1. 饿汉式

public class Singleton{

private static final Singleton instance=new Singleton();

private Singleton(){}

Public static Singleton getInstance(){

Return instance;

}

}

优点   
    1.线程安全   
    2.在类加载的同时已经创建好一个静态对象，调用时反应速度快   
缺点   
    资源效率不高，可能getInstance()永远不会执行到，但执行该类的其他静态方法或者加载了该类（class.forName)，那么这个实例仍然初始化

* 1. 懒汉式：

单例实例在第一次被使用时构建，延迟初始化。

class Test {

        private Test() {

        }

        public static Test instance = null;

        public static Test getInstance() {

                if (instance == null) {

              //多个线程判断instance都为null时，在执行new操作时多线程会出现重复情况

                        instance = new Singleton2();

                }

                return instance;

        }

}

优点：   
    避免了饿汉式的那种在没有用到的情况下创建事例，资源利用率高，不执行getInstance()就不会被实例，可以执行该类的其他静态方法。   
缺点：   
    懒汉式在单个线程中没有问题，但多个线程同事访问的时候就可能同事创建多个实例，而且这多个实例不是同一个对象，虽然后面创建的实例会覆盖先创建的实例，但是还是会存在拿到不同对象的情况。解决这个问题的办法就是加锁synchonized，第一次加载时不够快，多线程使用不必要的同步开销大。 

* 1. 静态内部类

class Test {

        private Test() {

        }

        private static class SingletonHelp {

                static Test instance = new Test();

        }

        public static Test getInstance() {

                return SingletonHelp.instance;

        }

}

优点   
    资源利用率高，不执行getInstance()不被实例，可以执行该类其他静态方法   
缺点   
    第一次加载时反应不够快

* 1. 枚举实现

public enum SingletonEnum{

INSTANCE;

public void doSomething(){

System.out.println(“do somthing”);

}

}

默认枚举实例的创建是线程安全的，即使**反序列化**也不会生成新的实例，**任何情况下都是一个单例**。

**优点：** 简单

* 1. DCL实现单例模式

public class Singleton {

private static Singleton instance = null;

private Singleton() { }

public static Singleton getInstance() {

// 两层判空，第一层是为了避免不必要的同步

//第二层是为了在null的情况下创建实例

if (instance == null) {

synchronized (Singleton.class)

{ if (instance == null) {

instance = new Singleton();

}

}

} return instance;

}

}

**优点：** 资源利用率高，既能够在需要的时候才初始化实例，又能保证线程安全，同时调用getInstance()方法**不进行同步锁**，效率高。  
**缺点：** 第一次加载时稍慢，由于**Java内存模型的原因偶尔会失败**。在高并发环境下也有一定的缺陷，虽然发生概率很小。  
DCL模式是使用最多的单例模式实现方式，除非代码在**并发场景比较复杂**或者**JDK 6以下版本**使用，否则，这种方式基本都能满足需求。

* 1. 容器实现单例

public class SingletonManager {

private static Map<String, Object> objMap = new HashMap<String, Object>();

public static void regsiterService(String key, Object instance) {

if (!objMap.containsKey(key)) {

objMap.put(key, instance);

}

}

public static Object getService(String key) {

return objMap.get(key);

}

}

SingletonManager可以**管理多个单例类型**，使用时根据key获取对象对应类型的对象。这种方式可以通过统一的接口获取操作，隐藏了具体实现，降低了耦合度。

1. 优缺点

优点：   
    1.在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就 防止其它对象对自己的实例化，确保所有的对象都访问一个实例   
    2.单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。   
    3.提供了对唯一实例的受控访问。   
    4.由于在系统内存中只存在一个对象，因此可以 节约系统资源，当 需要频繁创建和销毁的对象时单例模式无疑可以提高系统的性能。   
    5.允许可变数目的实例。   
    6.避免对共享资源的多重占用。   
缺点：   
    1.不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。   
    2.由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。   
    3.单例类的职责过重，在一定程度上违背了“单一职责原则”。   
    4.滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。

1. 场景

单例模式只允许创建一个对象，因此节省内存，加快对象访问速度，因此对象需要被公用的场合适合使用，如多个模块使用同一个数据源连接对象等等。如：   
    1.需要频繁实例化然后销毁的对象。   
    2.创建对象时耗时过多或者耗资源过多，但又经常用到的对象。   
    3.有状态的工具类对象。   
    4.频繁访问数据库或文件的对象。   
  
应用场景举例：   
    1.外部资源：每台计算机有若干个打印机，但只能有一个PrinterSpooler，以避免两个打印作业同时输出到打印机。内部资源：大多数软件都有一个（或多个）属性文件存放系统配置，这样的系统应该有一个对象管理这些属性文件   
    2. Windows的Task Manager（任务管理器）就是很典型的单例模式（这个很熟悉吧），想想看，是不是呢，你能打开两个windows task manager吗？ 不信你自己试试看哦~   
    3. 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。   
    4. 网站的计数器，一般也是采用单例模式实现，否则难以同步。   
    5. 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。   
    6. Web应用的配置对象的读取，一般也应用单例模式，这个是由于配置文件是共享的资源。   
    7. 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。数据库软件系统中使用数据库连接池，主要是节省打开或者关闭数据库连接所引起的效率损耗，这种效率上的损耗还是非常昂贵的，因为何用单例模式来维护，就可以大大降低这种损耗。   
    8. 多线程的线程池的设计一般也是采用单例模式，这是由于线程池要方便对池中的线程进行控制。   
    9. 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。   
    10. HttpApplication 也是单位例的典型应用。熟悉ASP.Net(IIS)的整个请求生命周期的人应该知道HttpApplication也是单例模式，所有的HttpModule都共享一个HttpApplication实例.

jdk：runtime

java.awt.Desktop

也用在其他设计模式中，抽象工厂模式、建造者模式、原型模式、外观模式等

1. 注意事项

使用注意事项：   
    1.使用时不能用反射模式创建单例，否则会实例化一个新的对象   
    2.使用懒单例模式时注意线程安全问题   
    3.饿单例模式和懒单例模式构造方法都是私有的，因而是不能被继承的，有些单例模式可以被继承（如登记式模式）

1. 参考

<http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples>

<http://www.javaworld.com/article/2073352/core-java/simply-singleton.html>

<https://sourcemaking.com/design_patterns/singleton>