singleton模式

单例模式，在系统里，你要判断一下，如果有一些类，只需要一个实例对象就可以了，那就给那个类做成单例的模式

实际上，我应该说，单例模式也是一个是个人就会的设计模式，因为我在面试的时候，单例模式，工厂模式。单例模式和工厂模式。太简单了。

常见的场景

（1）比如说，你自定义了一个框架，自定义了一份xml格式的一个配置文件，你要读取这个配置文件，这个配置文件中的数据，读取到类中，这个类的实例，只要保存一份就可以。那么此时可以使用单例模式，将这个类做成他的实例只能有一个，在这个实例中保存了配置文件中的数据

（2）类似于我们之前给大家讲解的那个工厂模式，有些工厂是需要实例化对象的，要基于实例化对象，来实现继承、接口、实现等功能，工厂实例可以做成单例的，就可以了

（3）你自己去判断，如果是一个类的实例只需要保持一份，那就做成单例

// 饿汉模式【提前 new 出来】

public class Singleton {

   private **static final** Singleton instance = new Singleton();

**private** Singleton() {

    // 不允许外部new

  }

   public static Singleton getInstance() {

       return instance;

  }

}

// 饱汉模式【第一次使用的时候才创建】--- 双重判断【还是不完美，因为不同的 JVM 编译器的问题，可能还是会线程不安全】

public class Singleton {

private static Singleton instance;

public static Singleton getInstance() {

       return instance;

  }

   public static Singleton getInstance() {

       if(instance == null) {

**synchronized**(Sinngleton.class) {

               if(instance == null) {

                   Singleton.instance = new Singleton();

              }

          }

      }

       return instance;

  }

}

// 【最最常见的单例模式】完全线程安全：用内部类

public class Singleton {

   private Singleton() {

  }

【内部类只要没有被使用，就不会初始化，实例就不会创建。且 JVM 能确保类静态初始化的过程**一定只会执行一次** ---- 只会初始化一次（静态内部类特点）】

**private static** class LazyHolder {

       private **static final** Singleton singleton = new Singleton();

  }

   public static Singleton getInstance() {

       return LazyHolder.singleton;

  }

}

最简单的一种类了，就是如果一个类就只需要一次，那么就使用这个类了

但是使用单例模式有一个要求，不允许这个类的逻辑过于复杂，一般就是持有某份配置文件的配置，或者是别的一些数据

因为如果别的很多类用了这个类，是没法打桩注入的，很麻烦

所以只能是简单的情况下，用单例模式，就是持有一份数据，但是这份数据全局就只要一份，比如说一些配置数据，就用单例模式，或者是类似redis的客户端实例，或者是类似elasticsearch的客户端实例

在咱们项目中，对一些工厂的实例，结合工厂模式来做，可以做成单例的

853769620