# 单例模式

## 模式是脱离语言的。

## 问题的由来：

为什么？

多个线程操作不同实例对象。多个线程要操作同一对象，要保证对象的唯一性。

## 解决的问题：

实例化过程中只实例化一次。

## 解决的思路

**有一个实例化的过程（只有一次），产生实例化对象。** new **对象**

**提供返回实例对象的方法。getInstace()**

## 单例模式的分类

3个维度：线程的安全性、性能、懒加载（lazy、延迟加载）。

#### 饿汉式

public class HungerySingleton {

//加载的时候就产生的实例对象

private static HungerySingleton instance=new HungerySingleton();

private HungerySingleton(){

}

//返回实例对象

public static HungerySingleton getInstance(){

return instance;

}

public static void main(String[] args) {

for (int i = 0; i < 20; i++) {

new Thread(()->{

System.out.println(HungerySingleton.getInstance());

}).start();

}

}

}

#### 线程安全性：JVM ClassLoader在加载的时候已经被实例化，所以只有这一次，线程安全的。

#### 懒加载：没有延迟加载，好长时间不使用，影响性能。

#### 性能比较好。

#### 懒汉式

public class HoonSingleton {

private static HoonSingleton instance=null;

private HoonSingleton(){

}

public static HoonSingleton getInstance(){

if(null==instance)

instance=new HoonSingleton();

return instance;

}

public static void main(String[] args) {

for (int i = 0; i < 20; i++) {

new Thread(()->{

System.out.println(HoonSingleton.getInstance());

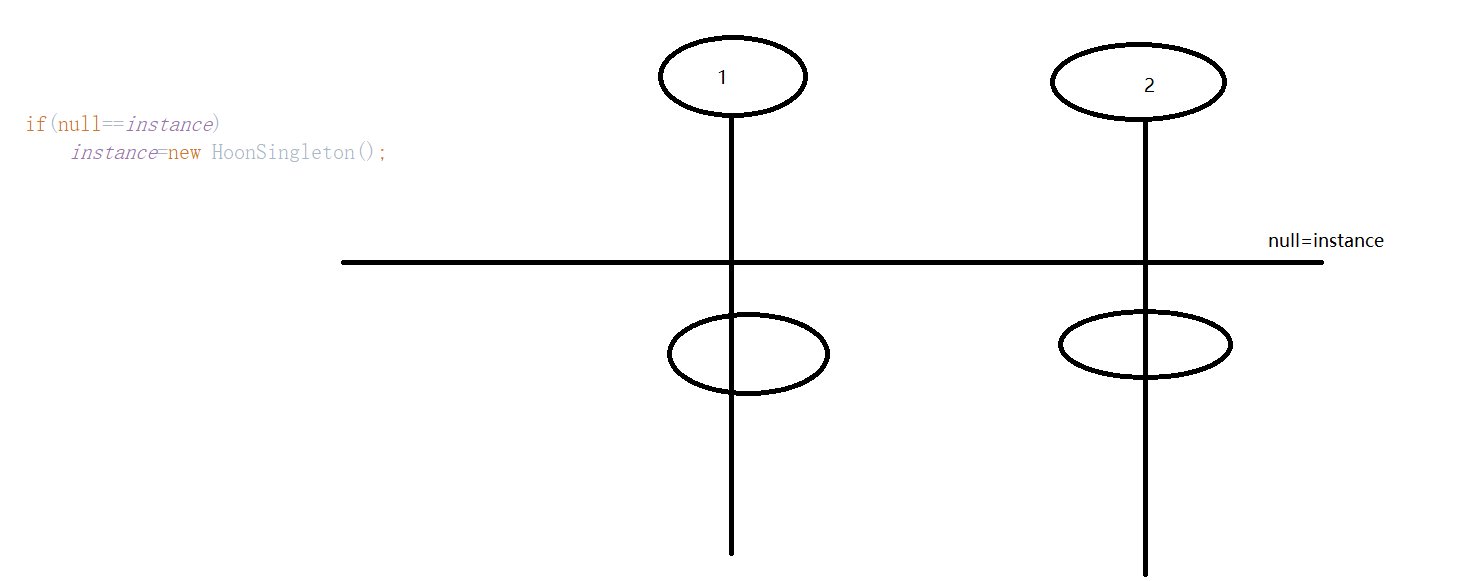
}).start();

}

}

}

#### 线程安全：不能保证实例对象的唯一性，懒加载，性能好。



#### 懒汉式+同步方法

线程安全：不安全

懒加载

性能：synchronized 退化到了串行执行。

#### Double-check-locking

#### public class DCL {

#### private static DCL instance=null;

#### private DCL(){

#### }

#### public static DCL getInstance(){

#### if(null==instance)

#### synchronized (DCL.class){

#### if(null==instance)

#### instance=new DCL();

#### }

#### return instance;

#### }

#### public static void main(String[] args) {

#### for (int i = 0; i < 20; i++) {

#### new Thread(()->{

#### System.out.println(DCL.getInstance());

#### }).start();

#### }

#### }

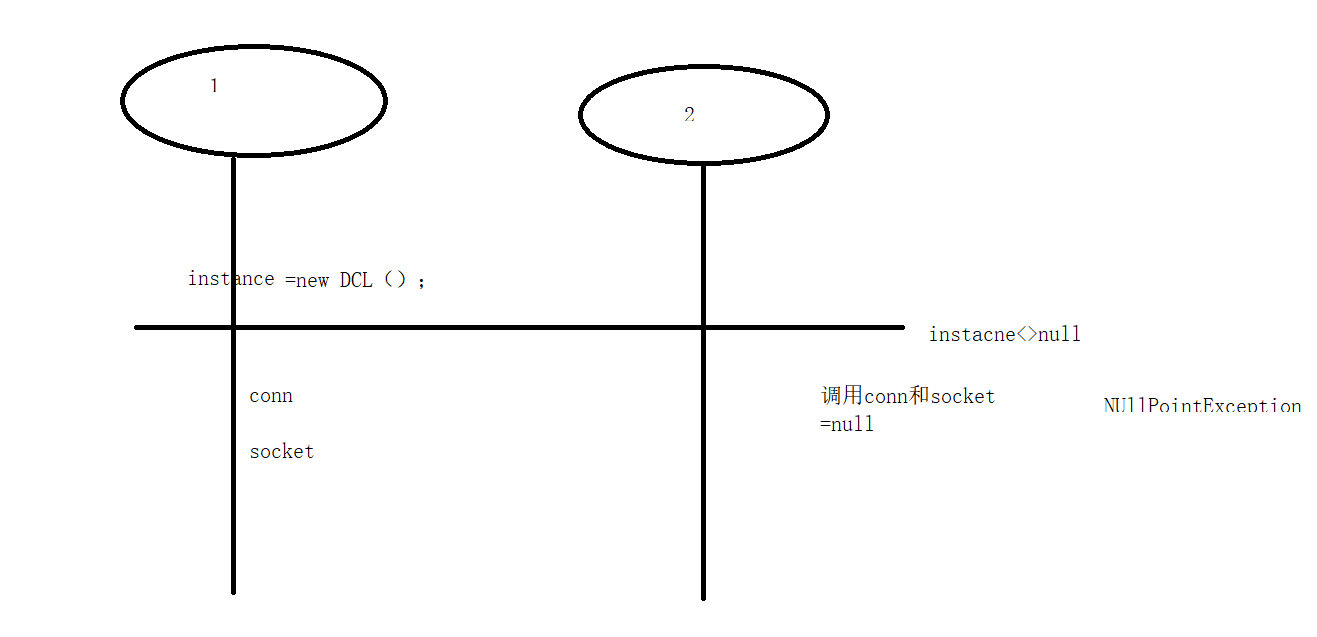
#### }

性能比较好

懒加载

线程的安全性

问题：因为指令重排一起空指针异常。



#### Volatile+Double-check

private volatile static DCL *instance*=null;

#### Holder

声明类的时候，成员变量中不声明实例变量，而放到内部静态类中

public class HolderDemo {

private HolderDemo(){

}

private static class Holder{

private static HolderDemo instance=new HolderDemo();

}

//懒加载

//synchronized

//<init>

public static HolderDemo getInstance(){

return Holder.instance;

}

//广泛的一种单例模式

}

#### 枚举

Effectice Java

public class EnumSingletonDemo {

private EnumSingletonDemo(){

}

//延迟加载

private enum EnumHolder{

INSTANCE;

private static EnumSingletonDemo instance=null;

private EnumSingletonDemo getInstance(){

instance=new EnumSingletonDemo();

return instance;

}

}//懒加载

public static EnumSingletonDemo getInstance(){

return EnumHolder.INSTANCE.instance;

}

}