# 单例模式

## 模式是脱离语言的。

## 问题的由来：

为什么要有单例模式？

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：

**1) 有一个实例化的过程（只有一次），产生实例化对象 new**

**2) 提供返回实例对象的方法 getInstance()**

## 单例模式的分类

线程的安全性、性能、懒加载（lazy ）

#### 饿汉式

#### <init>

*/\*\**

*\* 单例模式：饿汉式  
 \* 加载类时就产生实例对象  
 \*/*public class HungrySingleton {  
  
 // 加载到jvm时，就产生了实例对象  
 private static HungrySingleton *instance* = new HungrySingleton();  
  
 private HungrySingleton(){  
  
 }  
  
 // 返回实例对象  
 public static HungrySingleton getInstance(){  
 return *instance*;  
 }  
  
 public static void main(String[] args) {  
  
 for(int i=0; i<20; i++){  
 new Thread(()->{  
 System.*out*.println(HungrySingleton.*getInstance*());  
 }).start();  
 }  
  
 }  
  
}

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

#### 懒加载：没有延迟加载，长时间不使用依然在内存中，占据内存，多了可能出现内存溢出异常，

#### 影响性能

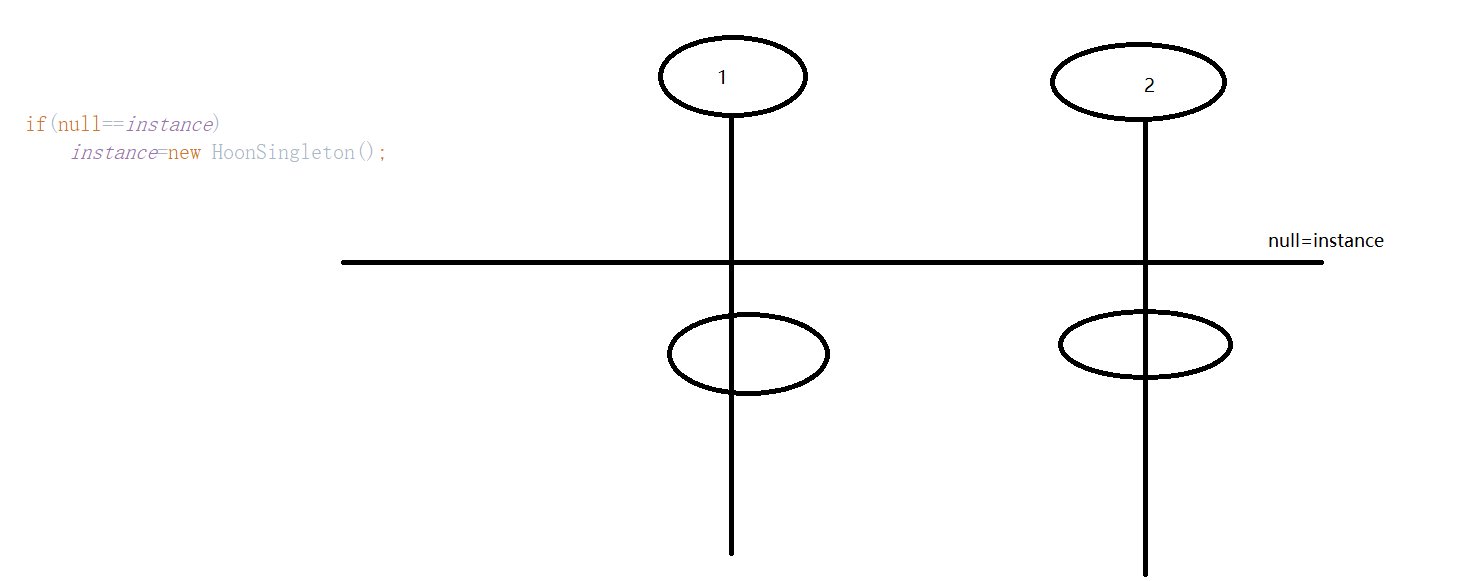
#### 性能比较好

#### 懒汉式

*/\*\**

*\* 单例模式：懒汉式  
 \* 在调用的过程中实例化对象  
 \*/*public class LazySingleton {  
  
 private static LazySingleton *instance* = null;  
  
 private LazySingleton(){ }  
  
 public static LazySingleton getInstance(){  
 if(null == *instance*)  
 *instance* = new LazySingleton();  
 return *instance*;  
 }  
  
 public static void main(String[] args) {  
 for (int i=0; i<20; i++){  
 new Thread(()->{  
 System.*out*.println(LazySingleton.*getInstance*());  
 }).start();  
 }  
 }  
}

#### 线程安全：不安全，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
* 懒加载
* 性能：使用synchronized关键字，退化到了串行执行

#### Double-Check-Locking

*/\*\**

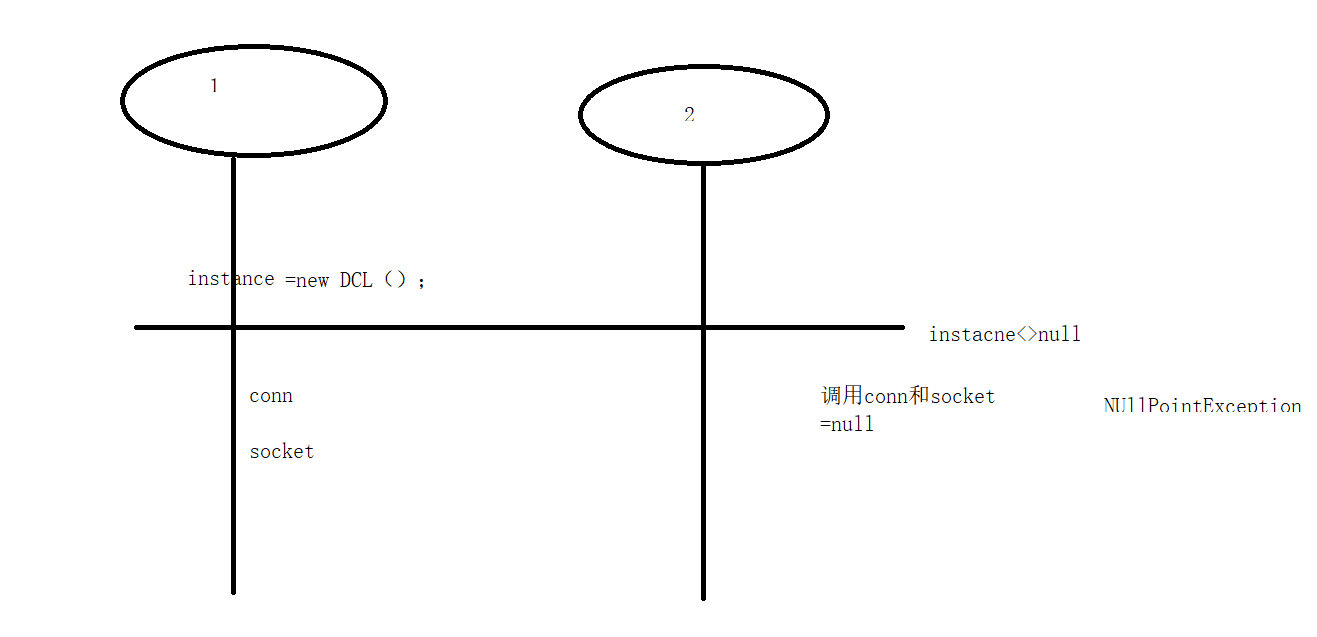
Double-Check-Locking *\** ***DC****L模式：双重枷锁  
 \*/*public class DCL {  
  
 private static DCL *instance* = null;  
 private DCL(){ }

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 模式

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，

*/\*\**

*\* Holder模式：目前应用比较广泛的单例模式  
 \*/*public class HolderDemo {  
  
 private HolderDemo(){ }  
  
 */\*\*  
 \* 定义内部类  
 \*/* private static class Holder{  
 private static HolderDemo *instance* = new HolderDemo();  
 }  
 //懒加载  
 //synchronized  
 //<init>  
  
 public static HolderDemo getInstance(){  
 return Holder.*instance*;  
 }  
}

#### 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;

}

}