# 饿汉式

在定义Singleton的时候就已经准备好了一个Singleton实例化对象INSTANCE,而并没有关心这个对象INSTANCE是否使用

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Singleton s = null;  s = Singleton.getInstance();  s.print();  }  }    class Singleton{  private static final Singleton INSTANCE = new Singleton();  private Singleton(){}  public void print(){  System.out.println("hello,world");  }  public static Singleton getInstance(){  return Singleton.INSTANCE;  }  } |

**说明:**

1)首先是构造函数必须私有化否则,可以实例化多个对象

Singleton s1 = new Singleton();

Singleton s2 = new Singleton();

2)既然是单例,则用static,所有实例共享一个对象,属性封装用private

3)然后定义一个返回static的属性的函数

# 懒汉式

需要实例的时候就创建,不过懒汉式存在线程安全的问题

|  |
| --- |
| class Singleton{  private static Singleton instance=null;  private Singleton(){}  public void print(){  System.out.println("hello,world");  }  public static Singleton getInstance(){  if(Singleton.instance==null){  instance = new Singleton();  }  return instance;  }  } |

解决方法1:

|  |
| --- |
| class Singleton{  private static Singleton instance=null;  private Singleton(){}  public void print(){  System.out.println("hello,world");  }  public static Singleton getInstance(){  synchronized("锁"){  if(Singleton.instance==null){  instance = new Singleton();  }  }  return instance;  }  public static void main(String args[]){  Singleton s = Singleton.getInstance();  }  } |

这种方式效率低

解决方式2:

|  |
| --- |
| class Singleton{  private static Singleton instance=null;  private Singleton(){}  public void print(){  System.out.println("hello,world");  }  public static Singleton getInstance(){  if(Singleton.instance == null){  synchronized("锁"){  if(Singleton.instance==null){  instance = new Singleton();  }  }  }  return instance;  }  public static void main(String args[]){  Singleton s = Singleton.getInstance();  }  } |