单例设计模式

一、什么是设计模式

1. **静态方法 和 静态属性 的经典使用.**
2. **大量的 实践总结 和 理论化之后优选的代码结构、代码风格、以及解决问题的思考方式.**

二、什么是单例模式

1. **采取一定的方法保证整个软件系统中，对某个类只能存在一个对象实例，并且该类只提供一个取得对其对象实例的方法.**
2. **单例模式有两种方式 （饿汉式 、 懒汉式）.**

饿汉式： （步骤）

1. **构造器私有化------>防止直接new.**
2. **类的内部创造对象.**
3. **向外暴露一个静态的公共方法.getinstance（获得实例）**
4. **代码实现.**

```
package com.single_;

public class SingleTon01 {
    public static void main(String[] args) {
//        GirlFriend xh = new GirlFriend("小红");
//        GirlFriend xb = new GirlFriend("xb");
        //静态方法可直接通过类直接获取
        GirlFriend instance = GirlFriend.getInstance();
        System.out.println(instance);
    }
}
//只能有一个女朋友
class GirlFriend{
    private String name;

    //如何保障我们只能创建一个对象？
    /*步骤:
    * 1.将构造器私有化
    * 2.在类的内部创建对象(该对象为static)
    * 3.提供一个公共的静态方法,返回gf对象*/
    //为了能够在静态方法中返回gf方法，需要static
    //这种对象通常是重量级对象(饿汉式可能造成创建了对象，但是没有用)
    private static GirlFriend gf = new GirlFriend("小红红");//在类内实例化唯一对象，加载类时就创建对象
    private GirlFriend(String name) {
        this.name = name;
    }
    public static GirlFriend getInstance(){//获取实例
```

```
        return gf;
    }

    @Override
    public String toString() {
        return "GirlFriend{" +
                "name='" + name + '\'' +
                '}';
    }
}
```

懒汉式:（步骤）

```
package com.single_;
/*懒汉式的单例模式*/
public class SingleTonO2 {
    public static void main(String[] args) {
//    由于构造器私有化所以创建不了对象      Cat tyd = new Cat("tyd");
        Cat instance = Cat.getInstance();//使用唯一对象时才创建对象
        System.out.println(instance);
    }
}
//只能创建一个对象
//使用单例模式

class Cat{
    private String name;
    /*懒汉式步骤:
     * 1.仍然构造器私有化
     * 2.定义一个静态属性对象
     * 3.提供一个public的static方法，返回一个Cat对象
     * 4.懒汉式，只有在用户使用getInstance时，才返回pc对象
     * 下次再调用会再返回上次的pc对象,因为pc不再是null了,保证了单例性*/
   /* private static Cat pc = new Cat("pechi");不要再new了，这种是饿汉式*/
    //懒汉式:
    private static Cat pc;//(2) 此时默认为null
    private Cat(String name) {//(1)
        this.name = name;
    }
    public static Cat getInstance(){//(3)
        if (pc == null) {
            pc = new Cat("ppc");//构造器会被调用一次，后面进不来就不再调用了，之后会直接返
回pc
        }
        return pc;
    }

    //pc被赋值得输出

    @Override
    public String toString() {
        return "Cat{" +
                "name='" + name + '\'' +
                '}';
    }
```

```
}
```

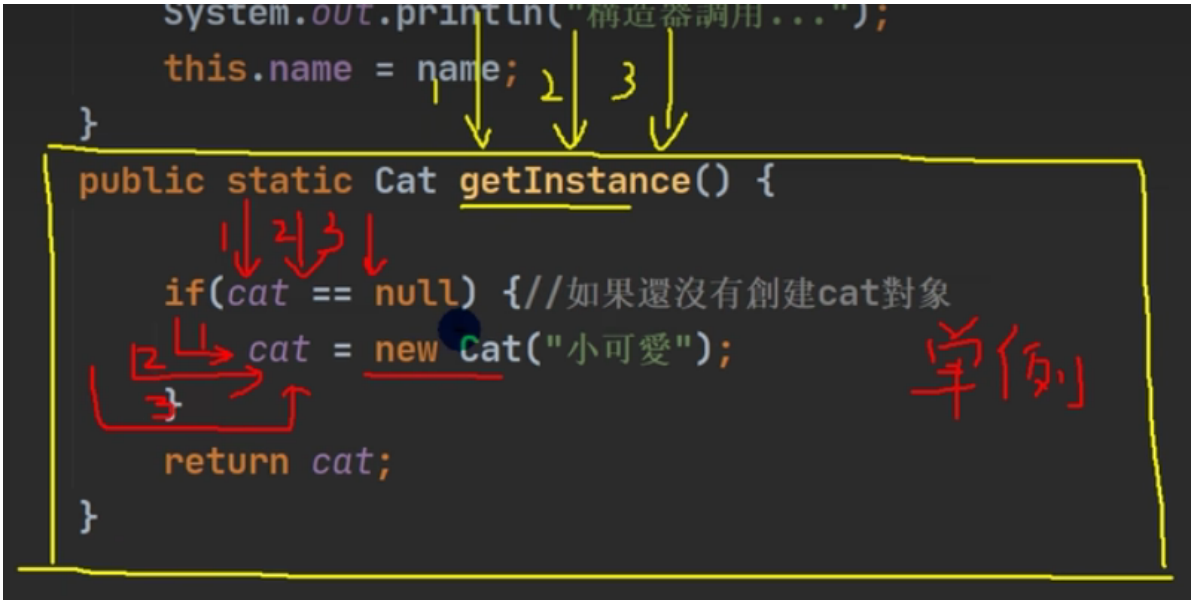
饿汉式 和 懒汉式 的区别.

- **创建对象时机不同.**

1. **饿汉式：是加载类时创建对象的.**
2. **懒汉式：是使用时才创建对象的.**

- **线程问题.**

1. **饿汉式不存在线程问题.**
2. **懒汉式存在线程问题.**

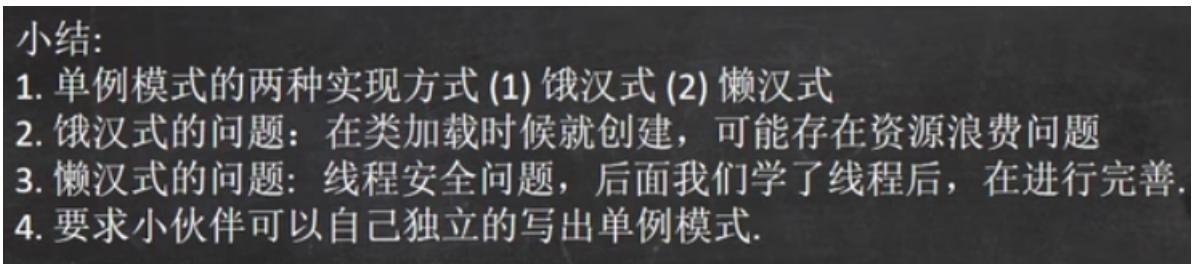


.

- **此时若三个线程统一时间点进来执行创建对象操作，单例性就被破坏了,被创造了三个相同对象.（后面详讲）**
- **内存浪费问题.**

1. **饿汉式：可能会造成空间浪费(因为加载类自然就创造了对象,用不到对象的化就浪费空间).**
2. **懒汉式:是要用到这个唯一对象的时候才会调用，相对更节省空间.**

小结:



。