**单例设计模式**

单例设计模式（多例设计模式）主要是一种控制实例化对象产生个数的设计；

单例设计:

|  |
| --- |
| class Siginleton {  public void print (){  System.out.println("www.gsdx.com");  }  }  public class JavaDemo{  public static void main(String[] args) {  Siginleton instanceA = new Siginleton();  Siginleton instanceB = new Siginleton();  Siginleton instanceC = new Siginleton();  **//此时可以产生无数个实例化对象**  instanceA.print() ;  instanceB.print() ;  instanceC.print() ;  }  } |

但是由于某些要求，我们现在要求Singleton 只给我们提供一个实例化对象。那么我们首先需要**控制的就是构造方法！**因为所有的新的实例化对象产生了，一定要调用构造方法，如果无法调用构造方法了，那么也就自然不能创建对象了。（private 属性可以让阻止外部的调用）

范例：构造方法私有化

|  |
| --- |
| class Singleton {  **private Singleton() {} //构造方法私有化**  public void print (){  System.out.println("www.gsdx.com");  }  }  public class JavaDemo{  public static void main(String[] args) {  Singleton instance=null ; //声明对象  instance = new Singleton(); //实例化对象  } |

那么按照正常的道理，此时该程序无法执行了。

无法new Singleton() ;

但是现在我们的需求是产生一个实例化对象。故上面的代码是不能满足要求的。分析上述代码:

1、private 访问权限的主要特点在于: 不能在类外部访问，但是可以类本身调用，所以我们可以考虑在类的内部调用。

范例: 在内部调用构造方法;

|  |
| --- |
| class Singleton {  **private Singleton instance = new Singleton() ;**  private Singleton() {} //构造方法私有化  public void print (){  System.out.println("www.gsdx.com");  }  } |

此时我们可以看到，我们在内部调用了构造方法，但是我们发现instance 是一个普通属性，而普通属性是在实例化对象产生之后才会被调用的，而这个时候外部无法产生实例化对象，因此这个属性也就无法被访问到了。

分析上面问题，我们知道必须在没有实例化的时候获取此属性！

那么就只有 static 属性可以成功了！

|  |
| --- |
| class Singleton {  **static** Singleton instance = new Singleton() ;  private Singleton() {} //构造方法私有化  public void print (){  System.out.println("www.gsdx.com");  }  }  public class JavaDemo{  public static void main(String[] args) {  Singleton instance = null ; //声明对象  instance = Singleton.instance ; //实例化对象  instance.print() ;  }  } |

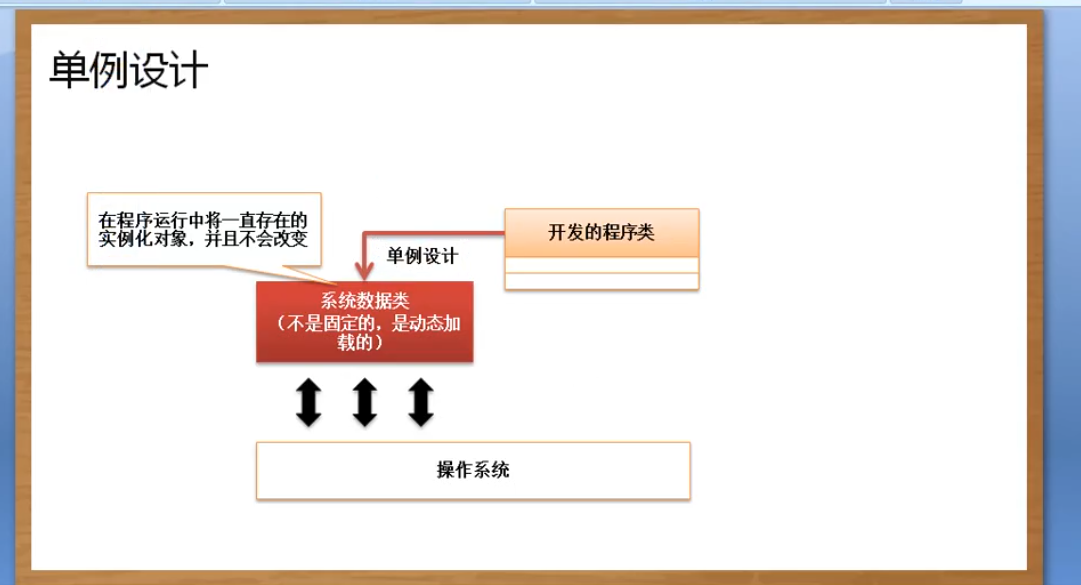
1. 我们在类种的属性，应该是封装后使用，所以理论上此时的instance 需要被封装，那么我们就可以用static 方法进行获取 instance

|  |
| --- |
| class Singleton {  private static Singleton instance = new Singleton() ;  **public static Singleton getInstance(){**  **return instance ;**  **}**  public void print (){  System.out.println("www.gsdx.com");  }  }  public class JavaDemo{  public static void main(String[] args) {  Singleton instance = null ; //声明对象  instance = Singleton.getInstance() ; //实例化对象  instance.print() ;  }  } |

我们一直在强调的是单例的设计模式也就是说控制实例化对象，但是这里我们依然可以创建多个对象。所以我们需要保证此时Singleton 类内部的instance 无法直接再次实例化。那么我们应该使用 final ,将INSTANCE 变为常量。

|  |
| --- |
| class Singleton {  private static **final** Singleton INSTANCE = new Singleton() ;  public static Singleton getInstance(){  return INSTANCE ;  }  public void print (){  System.out.println("www.gsdx.com");  }  }  public class JavaDemo{  public static void main(String[] args) {  Singleton instance = null ; //声明对象  instance = Singleton.getInstance() ; //实例化对象  instance.print() ;  }  } |

在很多情况下，是不需要重复产生对象的，例如:如果一个程序启动，那么肯定需要一个类负责保存程序加载的数据信息。



对于单例设计模式:，我们有两种模式：懒汉式，饿汉式。在之前定义的都是饿汉式。在系统加载时会自动提供Singleton 类的实例化，而还有一种懒汉式，就是在第一次使用的时候创建实例化对象。

范例：将上面的代码改为 懒汉式

|  |
| --- |
| class Singleton {  private static Singleton instance ; // final 去除  public static Singleton getInstance(){  if (instance == null ) { //第一次使用  instance = new Singleton(); //实例化对象  }  return instance ;  }  public void print (){  System.out.println("www.gsdx.com");  }  }  public class JavaDemo1{  public static void main(String[] args) {  Singleton instance = null ; //声明对象  instance = Singleton.getInstance() ; //实例化对象  instance.print() ;  }  } |

面试题: 请编写上面的一个Singleton 程序，并说明其主要特点；

·代码如上，可以把懒汉式（后面考虑到线程同步问题）和饿汉式都写上。

·特点: 构造方法私有化，类内部提供static 方法获取 实例化对象，这样不管外部如何操作永远只提供一个实例化对象。