

**编制：竹子**

Java Design pattern

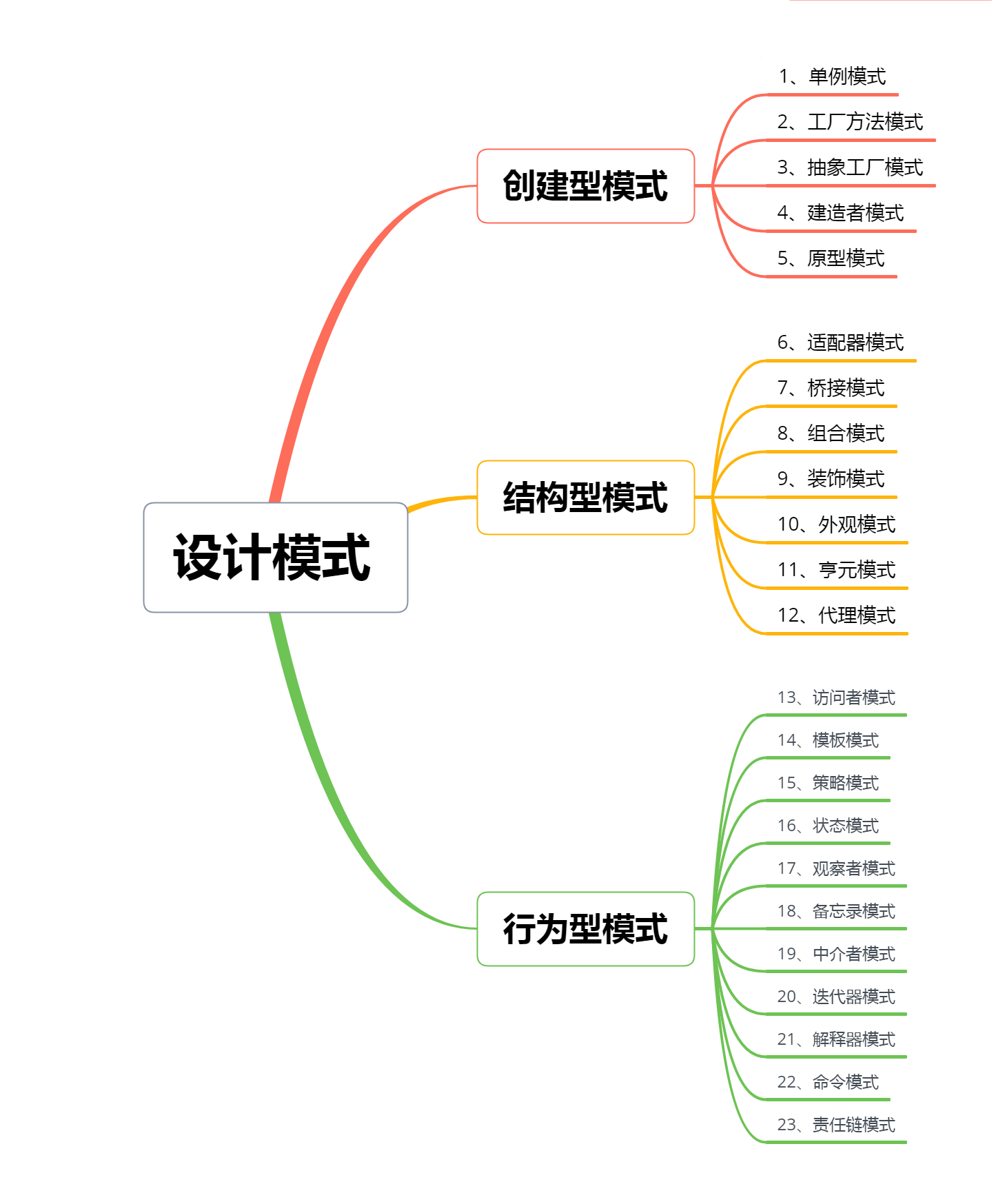
**设计模式**

# **一、设计模式分类**

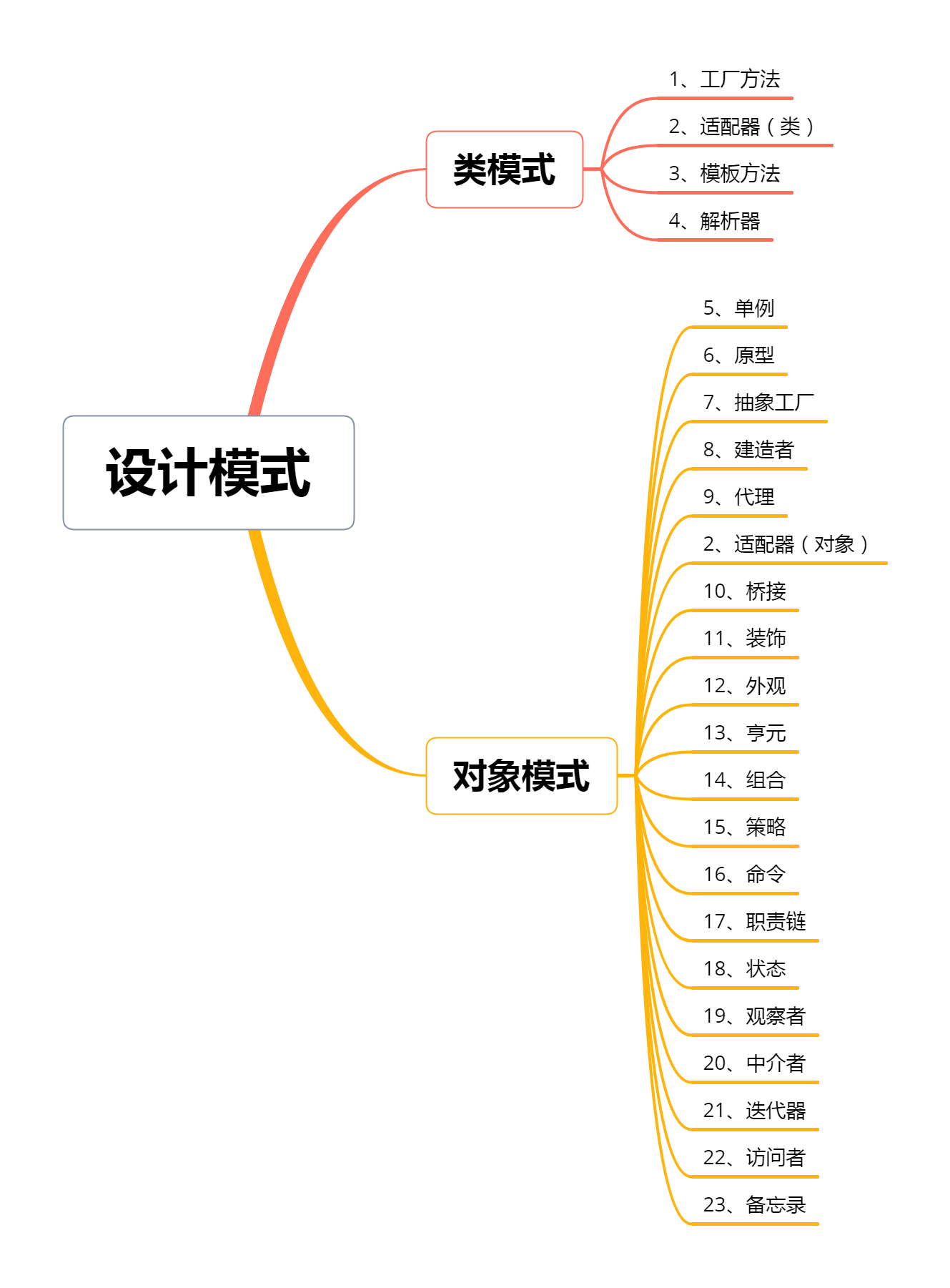
## **1.1.GOF分类表格**

|  |  |  |  |
| --- | --- | --- | --- |
| 范围\目的 | 创建型模式 | 构造型模式 | 行为型模式 |
| 类模式 | 工厂方法 | 适配器（类） | 模板方法  解析器 |
| 对象模式 | 单例  原型  抽象工厂  建造者 | 代理  适配器（对象）  桥接  装饰  外观  亨元  组合 | 策略  命令  职责链  状态  观察者  中介者  迭代器  访问者  备忘录 |

## **1.2.根据目的分类**



## **1.3.根据范围分类**



# **二、定义和应用**

## **2.1.单例模式**

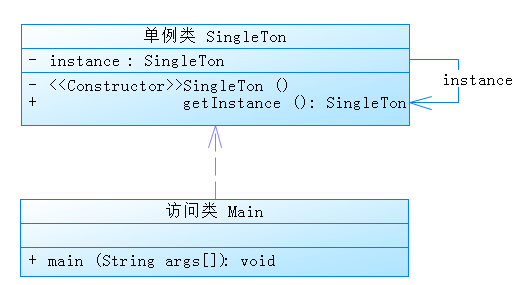
**定义**：某个类只能有一个实例，并提供一个全局的访问点获取该实例。

**特点：**1、只有一个实例对象。 2、自我实例化。 3、提供全局访问点。

**举例：**Windows 中只能打开一个任务管理器

**实现：**单例模式是设计模式中最简单的模式之一。通常，普通类的构造函数是公有的，外部类可以通过“new 构造函数()”来生成多个实例。但是，如果将类的构造函数设为私有的，外部类就无法调用该构造函数，也就无法生成多个实例。这时该类自身必须定义一个静态私有实例，并向外提供一个静态的公有函数用于创建或获取该静态私有实例。

**类图：**



### 2.1.1.第一种写法：懒汉式线程不安全单例模式

#### 2.1.1.1.单例类代码

|  |
| --- |
| 单例类 |
| /\*\*  \* 懒汉式线程不安全单例模式  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class LazySingleTon** {   /\*\* 1.只有一个实例对象 \*/  **private static LazySingleTon instance**;   /\*\* 2.自我实例化 \*/  **private LazySingleTon**() {  **if** (**instance** != **null**) {  **throw new** IllegalStateException(**"实例已经初始化！"**);  }  }  /\*\* 3.提供全局访问点 \*/  **public static LazySingleTon** getInstance() {  **if** (**instance** == **null**) {  **instance** = **new** LazySingleTon();  }  **return instance**;  } } |

#### 2.1.1.2.访问类代码

|  |
| --- |
| 访问类 |
| /\*\*  \* 访问类  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class Main** {   **public static void** main(**String**[] *args*) {  **LazySingleTon** ton1 = **LazySingleTon**.**getInstance**();  **System**.***out***.println(ton1);  **LazySingleTon** ton2 = **LazySingleTon**.**getInstance**();  **System**.***out***.println(ton2);  **System**.***out***.println(ton1 == ton2);  **System**.***out***.println(ton2.equals(ton1));  } } |

#### 2.1.1.3.说明

为什么说这是懒汉式呢？

因为实例对象只有在获取的时候才确定要不要实例化。

为什么说这是线程不安全呢？

因为提供实例获取的方法中，只是用了if (instance == null) 来确定要不要实例化，假设有两个线程去访问该方法，有可能会存在都执行instance = new LazySingleTon(); 这一句，所有就不能保证“只有一个实例对象”的原则了。所以说线程不安全。生产中建议不要使用“懒汉式线程不安全单例模式”。

### 2.1.2.第二种写法：懒汉式线程安全单例模式

#### 2.1.2.1.单例类代码

|  |
| --- |
| 单例类 |
| /\*\*  \* 懒汉式线程安全单例模式  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class LazySingleTonSafe** {  /\*\* 1.只有一个实例对象 \*/  **private static LazySingleTonSafe instance**;  /\*\* 2.自我实例化 \*/  **private LazySingleTonSafe**() {  **if** (**instance** != **null**) {  **throw new** IllegalStateException(**"实例已经初始化！"**);  }  }  /\*\* 3.提供全局访问点 \*/  **public static synchronized LazySingleTonSafe** getInstance() {  **if** (**instance** == **null**) {  **instance** = **new** LazySingleTonSafe();  }  **return instance**;  } } |

#### 2.1.2.2.访问类代码

|  |
| --- |
| 访问类类 |
| /\*\*  \* 访问类  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class Main** {   **public static void** main(**String**[] *args*) {  **LazySingleTonSafe** ton1 = **LazySingleTonSafe**.**getInstance**();  **System**.***out***.println(ton1);  **LazySingleTonSafe** ton2 = **LazySingleTonSafe**.**getInstance**();  **System**.***out***.println(ton2);  **System**.***out***.println(ton1 == ton2);  **System**.***out***.println(ton2.equals(ton1));  } } |

#### 2.1.2.3.说明

在“懒汉式线程不安全单例模式”基础上，在获取实例对象的方法上加了synchronized。

避免了线程冲突，保持单个对象。

### 2.1.3.第三种写法：双重检查懒汉式单例模式

#### 2.1.3.1.单例类代码

|  |
| --- |
| 单例类 |
| /\*\*  \* 双重检查懒汉式单例模式  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class DoubleCheckedLockLazySingleTon** {  /\*\* 1.只有一个实例对象 \*/  **private static volatile DoubleCheckedLockLazySingleTon instance**;  /\*\* 2.自我实例化 \*/  **private DoubleCheckedLockLazySingleTon**() {  **if** (**instance** != **null**) {  **throw new** IllegalStateException(**"实例已经初始化！"**);  }  }  /\*\* 3.提供全局访问点 \*/  **public static DoubleCheckedLockLazySingleTon** getInstance() {  **if** (**instance** == **null**) {  **synchronized** (**DoubleCheckedLockLazySingleTon**.**class**) {  **if** (**instance** == **null**) {  **instance** = **new** DoubleCheckedLockLazySingleTon();  }  }  }  **return instance**;  } } |

#### 2.1.3.2.访问类代码

|  |
| --- |
| 访问类类 |
| /\*\*  \* 访问类  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class Main** {   **public static void** main(**String**[] *args*) {  **DoubleCheckedLockLazySingleTon** ton1 = **DoubleCheckedLockLazySingleTon**.**getInstance**();  **System**.***out***.println(ton1);  **DoubleCheckedLockLazySingleTon** ton2 = **DoubleCheckedLockLazySingleTon**.**getInstance**();  **System**.***out***.println(ton2);  **System**.***out***.println(ton1 == ton2);  **System**.***out***.println(ton2.equals(ton1));  } } |

#### 2.1.3.3.说明

在“懒汉式线程不安全单例模式”基础上，在属性上加上volatile，

和在获取实例对象的方法上加了synchronized。

避免了线程冲突，保持单个对象。

### 2.1.4.第四种写法：饿汉式单例模式

#### 2.1.4.1.单例类代码

|  |
| --- |
| 单例类 |
| /\*\*  \* 饿汉式单例模式  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class HungrySingleTon** {   /\*\* 1.只有一个实例对象 \*/  **private static final HungrySingleTon *instance*** = **new** HungrySingleTon();  /\*\* 2.自我实例化 \*/  **private HungrySingleTon**() {  **if** (***instance*** != **null**) {  **throw new** IllegalStateException(**"实例已经初始化！"**);  }  }  /\*\* 3.提供全局访问点 \*/  **public static HungrySingleTon** getInstance() {  **return *instance***;  } } |

#### 2.1.4.2.访问类代码

|  |
| --- |
| 访问类类 |
| /\*\*  \* 访问类  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class Main** {   **public static void** main(**String**[] *args*) {  **HungrySingleTon** ton1 = **HungrySingleTon** .**getInstance**();  **System**.***out***.println(ton1);  **HungrySingleTon** ton2 = **HungrySingleTon** .**getInstance**();  **System**.***out***.println(ton2);  **System**.***out***.println(ton1 == ton2);  **System**.***out***.println(ton2.equals(ton1));  } } |

#### 2.1.4.3.说明

### 2.1.5.第五种写法：静态内部类方式单例模式

#### 2.1.5.1.单例类代码

|  |
| --- |
| 单例类 |
| /\*\*  \* 静态内部类方式单例模式  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class StaticInnerClassSingleTon** {   /\*\* 2.自我实例化 \*/  **private StaticInnerClassSingleTon**() { }  /\*\* 3.提供全局访问点 \*/  **public static StaticInnerClassSingleTon** getInstance() {  **return StaticInnerClassSingleTonHolder**.***instance***;  }  /\*\* 1.只有一个实例对象 \*/  **private static class StaticInnerClassSingleTonHolder** {  **private static final StaticInnerClassSingleTon *instance*** = **new** StaticInnerClassSingleTon();  } } |

#### 2.1.5.2.访问类代码

|  |
| --- |
| 访问类类 |
| /\*\*  \* 访问类  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class Main** {   **public static void** main(**String**[] *args*) {  **StaticInnerClassSingleTon** ton1 = **StaticInnerClassSingleTon** .**getInstance**();  **System**.***out***.println(ton1);  **StaticInnerClassSingleTon** ton2 = **StaticInnerClassSingleTon** .**getInstance**();  **System**.***out***.println(ton2);  **System**.***out***.println(ton1 == ton2);  **System**.***out***.println(ton2.equals(ton1));  } } |

#### 2.1.5.3.说明

### 2.1.6.第六种写法：枚举类型实现单例模式

#### 2.1.6.1.单例类代码

|  |
| --- |
| 单例类 |
| /\*\*  \* 枚举类型实现单例模式  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class StaticInnerClassSingleTon** {   /\*\* 2.自我实例化 \*/  **private StaticInnerClassSingleTon**() { }  /\*\* 3.提供全局访问点 \*/  **public static StaticInnerClassSingleTon** getInstance() {  **return StaticInnerClassSingleTonHolder**.***instance***;  }  /\*\* 1.只有一个实例对象 \*/  **private static class StaticInnerClassSingleTonHolder** {  **private static final StaticInnerClassSingleTon *instance*** = **new** StaticInnerClassSingleTon();  } } |

#### 2.1.6.2.访问类代码

|  |
| --- |
| 访问类类 |
| /\*\*  \* 访问类  \* **@author** 竹子  \* **@version** 1.0  \*/ **public class Main** {   **public static void** main(**String**[] *args*) {  **StaticInnerClassSingleTon** ton1 = **StaticInnerClassSingleTon** .**getInstance**();  **System**.***out***.println(ton1);  **StaticInnerClassSingleTon** ton2 = **StaticInnerClassSingleTon** .**getInstance**();  **System**.***out***.println(ton2);  **System**.***out***.println(ton1 == ton2);  **System**.***out***.println(ton2.equals(ton1));  } } |

#### 2.1.6.3.说明