# 设计模式

* 分类：设计模式分为三种类型，共23种

1. 创建型模式：
2. 结构型模式：
3. 行为型模式：

# 单例模式

## 定义：对某个类只存在一个实例对象

## 实现方式：

### 饿汉式（静态常量）

优缺点：

缺点:内存浪费

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton;  **import** java.security.Signature;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 15:11  \*/* **public class** SingletonTest01 {  **public static void** main(String[] args) {  Singleton instance = Singleton.*getInstance*();  Singleton instance2 = Singleton.*getInstance*();  System.***out***.println(instance == instance2);  System.***out***.println(**"instance.hashCode="** + instance.hashCode());  System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  } }  *//饿汉式* **class** Singleton{   *//1.构造器私有化，外部能new* **private** Singleton(){   }  *//2.本类内部创建对象实例* **private final static** Singleton ***instance*** = **new** Singleton();   *//3.提供一个公有的静态方法，返回实例对象* **public static** Singleton getInstance(){  **return *instance***;  } } |

### 饿汉式（静态代码块）

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton.type2;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 15:11  \*/* **public class** SingletonTest02 {  **public static void** main(String[] args) {  Singleton instance = Singleton.*getInstance*();  Singleton instance2 = Singleton.*getInstance*();  System.***out***.println(instance == instance2);  System.***out***.println(**"instance.hashCode="** + instance.hashCode());  System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  } }  *//饿汉式* **class** Singleton{     *//1.构造器私有化，外部能new* **private** Singleton(){   }  *//2.本类内部创建对象实例* **private static** Singleton *instance*;   **static** { *//静态代码块中，创建单例对象  instance* = **new** Singleton();  }   *//3.提供一个公有的静态方法，返回实例对象* **public static** Singleton getInstance(){  **return** *instance*;  } } |

### 懒汉式（线程不安全）

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton.type3;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 15:23  \*/* **public class** SingletonTest03 {  **public static void** main(String[] args) {  System.***out***.println(**"懒汉式线程不安全"**);  Singleton instance = Singleton.*getInstance*();  Singleton instance2 = Singleton.*getInstance*();  System.***out***.println(instance == instance2);  System.***out***.println(**"instance.hashCode="** + instance.hashCode());  System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  } }  **class** Singleton{   **private static** Singleton *instance*;   **private** Singleton(){   }   *//提供一个静态的公有方法，当使用到该方法时，才去创建instance  //即懒汉式* **public static** Singleton getInstance(){  **if**(*instance* == **null**){  *instance* = **new** Singleton();  }  **return** *instance*;  } } |

### 懒汉式（线程安全，同步方法）

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton.type4;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 15:28  \*/* **public class** SingletonTest04 {  **public static void** main(String[] args) {  System.***out***.println(**"懒汉式2，线程安全~"**);  Singleton instance = Singleton.*getInstance*();  Singleton instance2 = Singleton.*getInstance*();  System.***out***.println(instance == instance2);  System.***out***.println(**"instance.hashCode="** + instance.hashCode());  System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  } }  **class** Singleton{   **private static** Singleton *instance*;   **private** Singleton(){   }   *//提供一个静态的公有方法，计入同步处理的代码，解决线程安全,当使用到该方法时，才去创建instance  //即懒汉式* **public static synchronized** Singleton getInstance(){  **if**(*instance* == **null**){  *instance* = **new** Singleton();  }  **return** *instance*;  } } |

### 懒汉式（线程不安全，同步代码块）

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton.type5;   */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 15:28  \*/* **public class** SingletonTest05 {  **public static void** main(String[] args) {  System.***out***.println(**"懒汉式2，线程安全~"**);  Singleton instance = Singleton.*getInstance*();  Singleton instance2 = Singleton.*getInstance*();  System.***out***.println(instance == instance2);  System.***out***.println(**"instance.hashCode="** + instance.hashCode());  System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  } }  **class** Singleton{   **private static** Singleton *instance*;   **private** Singleton(){   }   *//提供一个静态的公有方法，计入同步处理的代码，解决线程安全,当使用到该方法时，才去创建instance  //即懒汉式* **public static** Singleton getInstance(){  **if**(*instance* == **null**){  **synchronized** (Singleton.**class**){  *instance* = **new** Singleton();  }  }  **return** *instance*;  } } |

### 双重检查

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton.type6;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 15:23  \*/* **public class** SingletonTest06 {  **public static void** main(String[] args) {  System.***out***.println(**"懒汉式线程不安全"**);  Singleton instance = Singleton.*getInstance*();  Singleton instance2 = Singleton.*getInstance*();  System.***out***.println(instance == instance2);  System.***out***.println(**"instance.hashCode="** + instance.hashCode());  System.***out***.println(**"instance2.hashCode="** + instance2.hashCode());  } }  **class** Singleton{   **private static** Singleton *instance*;   **private** Singleton(){   }   *//提供一个静态的公有方法，当使用到该方法时，才去创建instance  //即懒汉式* **public static** Singleton getInstance(){  **if**(*instance* == **null**){  **synchronized** (Singleton.**class**) {  **if**(*instance* == **null**){  *instance* = **new** Singleton();  }  }  }  **return** *instance*;  } } |

### 静态内部类

静态内部类的特点：（懒加载、线程安全）

当singleton进行类装载的时候，静态内部类是不会被装载的

当静态内部类调用getInstance的时候，静态内部类被装载，而且只会装载一次

### 枚举

|  |
| --- |
| **package** com.tracy.pattern.com.tracy.pattern.singleton.type8;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 17:38  \*/* **public class** SingletonTest08 {  **public static void** main(String[] args) {  Singleton instance = Singleton.***INSTANCE***;  Singleton instance2 = Singleton.***INSTANCE***;  System.***out***.println(instance == instance2);  System.***out***.println(instance.hashCode());  System.***out***.println(instance2.hashCode());  } }  *//使用枚举，可以实现单例* **enum** Singleton{  ***INSTANCE***; *//属性* **public void** sayOK(){  System.***out***.println(**"ok"**);  } |

### 推荐使用

枚举、静态内部类、双重检查、饿汉式(确定实例会被使用)

## JDK源码

|  |
| --- |
| **public class** Runtime {  **private static** Runtime *currentRuntime* = **new** Runtime();   */\*\*  \* Returns the runtime object associated with the current Java application.  \* Most of the methods of class <code>Runtime</code> are instance  \* methods and must be invoked with respect to the current runtime object.  \*  \** ***@return*** *the <code>Runtime</code> object associated with the current  \* Java application.  \*/* **public static** Runtime getRuntime() {  **return** *currentRuntime*;  }   */\*\* Don't let anyone else instantiate this class \*/* **private** Runtime() {}  }  } |

# 适配器模式