Java 单例模式讲解和代码示例

单例是一种创建型设计模式，让你能够保证一个类只有一个实例，并提供一个访问该实例的全局节点。

单例拥有与全局变量相同的优缺点。尽管它们非常有用，但却会破坏代码的模块化特性。

在某些其他上下文中，你不能使用依赖于单例的类。你也将必须使用单例类。绝大多数情况下，该限制会在创建单元测试时出现。

进一步了解单例模式 →

在 Java 中使用模式

复杂度： ★☆☆

流行度： ★★☆

使用示例：许多开发者将单例模式视为一种反模式。因此它在 Java 代码中的使用频率正在逐步减少。

尽管如此，Java 核心程序库中仍有相当多的单例示例：

- `java.lang.Runtime#getRuntime()`
- `java.awt.Desktop#getDesktop()`
- `java.lang.System#getSecurityManager()`

识别方法：单例可以通过返回相同缓存对象的静态构建方法来识别。

导航

简介

基础单例（单线程）
Singleton
DemoSingleThread
OutputDemoSingleThread

基础单例（多线程）
Singleton
DemoMultiThread
OutputDemoMultiThread

采用延迟加载的线程安全单例
Singleton
DemoMultiThread
OutputDemoMultiThread

希望了解更多?

基础单例（单线程）

实现一个粗糙的单例非常简单。你仅需隐藏构造函数并实现一个静态的构建方法即可。

Singleton.java: 单例

```
package refactoring_guru.singleton.example.non_thread_safe;

public final class Singleton {
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // The following code emulates slow initialization.
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
```

```
        this.value = value;
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            instance = new Singleton(value);
        }
        return instance;
    }
}
```

DemoSingleThread.java: 客户端代码

```
package refactoring_guru.singleton.example.non_thread_safe;

public class DemoSingleThread {
    public static void main(String[] args) {
        System.out.println("If you see the same value, then singleton was reused (yay!)" + "\n
                "If you see different values, then 2 singletons were created (booo!!)" + "\n\n
                "RESULT:" + "\n");
        Singleton singleton = Singleton.getInstance("FOO");
        Singleton anotherSingleton = Singleton.getInstance("BAR");
        System.out.println(singleton.value);
        System.out.println(anotherSingleton.value);
    }
}
```

OutputDemoSingleThread.txt: 执行结果

```
If you see the same value, then singleton was reused (yay!)
If you see different values, then 2 singletons were created (booo!!)

RESULT:

FOO
FOO
```

基础单例（多线程）

相同的类在多线程环境中会出错。多线程可能会同时调用构建方法并获取多个单例类的实例。

Singleton.java: 单例

```
package refactoring_guru.singleton.example.non_thread_safe;

public final class Singleton {
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // The following code emulates slow initialization.
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        this.value = value;
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            instance = new Singleton(value);
        }
        return instance;
    }
}
```

DemoMultiThread.java: 客户端代码

```
package refactoring_guru.singleton.example.non_thread_safe;

public class DemoMultiThread {
    public static void main(String[] args) {
        System.out.println("If you see the same value, then singleton was reused (yay!)" + "\n
                "If you see different values, then 2 singletons were created (booo!!)" + "\n\n
                "RESULT:" + "\n");
        Thread threadFoo = new Thread(new ThreadFoo());
        Thread threadBar = new Thread(new ThreadBar());
        threadFoo.start();
        threadBar.start();
    }

    static class ThreadFoo implements Runnable {
        @Override
        public void run() {
            Singleton singleton = Singleton.getInstance("FOO");
            System.out.println(singleton.value);
        }
    }

    static class ThreadBar implements Runnable {
```

```
        @Override
        public void run() {
            Singleton singleton = Singleton.getInstance("BAR");
            System.out.println(singleton.value);
        }
    }
}
```

OutputDemoMultiThread.txt: 执行结果

```
If you see the same value, then singleton was reused (yay!)
If you see different values, then 2 singletons were created (booo!!)

RESULT:

FOO
BAR
```

采用延迟加载的线程安全单例

为了解决这个问题，你必须在创建首个单例对象时对线程进行同步。

Singleton.java: 单例

```
package refactoring_guru.singleton.example.thread_safe;

public final class Singleton {
    // The field must be declared volatile so that double check lock would work
    // correctly.
    private static volatile Singleton instance;

    public String value;

    private Singleton(String value) {
        this.value = value;
    }

    public static Singleton getInstance(String value) {
        // The approach taken here is called double-checked locking (DCL). It
        // exists to prevent race condition between multiple threads that may
        // attempt to get singleton instance at the same time, creating separate
        // instances as a result.
        //
        // It may seem that having the `result` variable here is completely
```

```
        // pointless. There is, however, a very important caveat when
        // implementing double-checked locking in Java, which is solved by
        // introducing this local variable.
        //
        // You can read more info DCL issues in Java here:
        // https://refactoring.guru/java-dcl-issue
        Singleton result = instance;
        if (result != null) {
            return result;
        }
        synchronized(Singleton.class) {
            if (instance == null) {
                instance = new Singleton(value);
            }
            return instance;
        }
    }
}
```

DemoMultiThread.java: 客户端代码

```
package refactoring_guru.singleton.example.thread_safe;

public class DemoMultiThread {
    public static void main(String[] args) {
        System.out.println("If you see the same value, then singleton was reused (yay!)" + "\n
                "If you see different values, then 2 singletons were created (booo!!)" + "\n\n
                "RESULT:" + "\n");
        Thread threadFoo = new Thread(new ThreadFoo());
        Thread threadBar = new Thread(new ThreadBar());
        threadFoo.start();
        threadBar.start();
    }

    static class ThreadFoo implements Runnable {
        @Override
        public void run() {
            Singleton singleton = Singleton.getInstance("FOO");
            System.out.println(singleton.value);
        }
    }

    static class ThreadBar implements Runnable {
        @Override
        public void run() {
            Singleton singleton = Singleton.getInstance("BAR");
            System.out.println(singleton.value);
        }
    }
}
```

OutputDemoMultiThread.txt: 执行结果

```
If you see the same value, then singleton was reused (yay!)
If you see different values, then 2 singletons were created (booo!!)

RESULT:

BAR
BAR
```

希望了解更多?

Java 中还有更多特殊类型的单例模式。阅读这篇文章以了解更多：

★ **Java 单例设计模式的最佳实践与示例**

继续阅读

Java 适配器模式讲解和代码示例 →

返回

← Java 原型模式讲解和代码示例

单例在其他编程语言中的实现

主页
重构
设计模式
会员专属内容
论坛
联系我们

Khmelnitske shosse 19 / 27, Kamianets-Podilskyi, 乌克兰, 32305
Email: support@refactoring.guru

图片作者： Dmitry Zhart