设计模式是什么鬼（单例）

凸凹 设计模式 2018年7月23日 1.66K 1 5

之前我们讲过面向对象以及封装、继承、多态三大特性，底子打好了那我们就把设计模式一个个拆开来看看到底都是神些什么鬼，我们先从简单的单例说起吧。单例，顾名思义，整个系统其实就只有一个实例存在，不能再多，否则就不叫单例。那我们把整个宇宙看做是一个庞大的系统，这宇宙里有各种对象存在，人啊，动物啊，植物啊不胜枚举，这些都是实例，丰富多彩的世界是美好的。然而，持续几千年的战争给世界带来了巨大灾难，尤其是宗教战争最为残忍，各个信仰间存在极大的世界观价值观冲突。

单印度一个国家就有几百个神，人们各信各的，风俗各异，各邦文化冲突不断，语言不通，办事效率极低。

为了让幸福美好洒满人间，那我们就定义一位神吧，独一无二的神。

我们先写一个God类吧，类中空空如也，世界如此清净，虚无缥缈。

```
1 public class God {
2
3 }
```

首先我们得保证任何人都不能去创建神的实例，否则如：new God()，这样世界又要陷入战争的灾难，各种造神

凸凹 官方
关注 私信
凸凹里歐，十余年以上开发经验，《设计
列文章作者，旨在用最通俗的方式诠释设

最新文章
设计模式是什么鬼（访问者）
设计模式是什么鬼（命令模式）
设计模式是什么鬼（建造者）
设计模式是什么鬼（抽象工厂）
设计模式是什么鬼（工厂方法）

极客快讯
传Android 要收费，周鸿祎回应
2019年1月4日 124
Python 3.7 上架微软商店
2019年1月3日 90
fish 3.0.0 发布
2019年1月3日 194
股价跳水、变现受阻，美图做中国下戏吗
2019年1月3日 66
中兴通讯5G核心网率先通过IMT202
2019年1月3日 46

```
public class God {
    private God(){}//构造方法私有化
}
```

God类里面封装一个God自己，对，一切都是神创造的，包括我们人类。有人开始质疑，那神是谁？神自己是谁造的？这是个哲学问题。神说“I am who I am.” 我是我所是，我就是我，自有永有，超越时空。很逆天吧？好吧，谁也不能造上帝，神自己造自己。

```
public class God {
    private static final God god = new God();//自有永有的神单例
    private God(){}//构造方法私有化
}
```

以上private关键字保证了上帝的私有性，不可见性，不可访问性，我想没有活人见过上帝吧？static关键字保证上帝的静态性，他与类同在，不依赖于类的实例化就自有永有，他将在内存中永生，GC垃圾回收器也回收不了他。final关键字则保证这位神是和常量，衡量，他是终极上帝，不能再改。

正如同静态方法main()，不需要实例化类就能运行的入口，同样我们需要一个静态方法getInstance()来请神，方法体内我们就返回这个在唯一的真神，当然方法它必须是public公开的，不然谁都访问不了。

```
public class God {
    private static final God god = new God();//自有永有的神单例
    private God(){}//构造方法私有化
    public static God getInstance(){//请神方法公开化
        return god;
    }
}
```

以上的神类雏形已经写好了，当然你还可以加其他的功能方法，比如说创世纪神造了光，造了世界、动物、人、亚当夏娃等等功能，我们这里就不在赘述了。那对于外部来说只要调用God.getInstance();就可以拿到神了，而且不管谁拿，拿几次，都是同一个神，这样就保证了整个系统中神的唯一性，不可伪造性，至于其他先知那也只是神的代理人，只能帮请神而已。

好了，其实我们已经学会了单例模式的“痴汉模式（Eager load）”，代码第一行一开始就造出了神（new God那一句），已经准备好了随时给你请神，这样就有了一个问题，如果没人请神那不是白造了？提前备货如果价格跌了不是很惨？反应在系统中的问题就是占用了内存空间。于是又有了“懒汉模式（Lazy load）”

```
public class God {
    private static God god;//这里不进行实例化
    private God(){}
    public static God getInstance() {
        if (god == null) {//如果无神才造神
            god = new God();
        }
        return god;
    }
}
```

这我们看到一开始就没有造神，只有某人第一次求神时才实例化，之后再求的就直接返回了。这样的好处是省了一段时间的内存（无求神期间），坏处是第一次请神的时候速度相较之前的痴汉模式会慢，因为要消耗CPU去造神。

其实这么写是在多线程模式下是有陷阱的，试想多人同时并发请神的话，依然会造成多神，好吧我们再来改良一下，把请神方法加上synchronized，声明为同步方法，某线程调用前必须获取同步锁，调用完后会释放锁给其他线程用，也就是请神的必须排队，大家一个一个按顺序来。

```
public class God {
    private static God god;//这里不进行实例化
    private God(){}
    public static synchronized God getInstance() {//此处加入同步
        if (god == null) {//如果无神才造神
            god = new God();
        }
        return god;
    }
}
```

然而，这样做是要付出代价的，还没进庙呢不管三七二十一请神的直接给加锁排队，结果队伍从北边的庙排到了南天门，人们都要来一个一个拜佛求神，这造成了巨大时间浪费，没有充分利用CPU资源并发优势（特别是多核情况）。好吧，那还是让人们抢好了，但依然得保证单例神的情况下。

这里我们去掉方法上的同步关键字，换到方法体内部做同步，整个方法开放并发大家都可以同时入庙，当然起早贪黑的虔诚信徒们要抢头香是必须要入堂排队的。一旦头香诞生，那其他抢香的都白早起，白排队了。再之后的事情我们都可以预见了，头注香被抢后堂内排队再无必要来了，大家可以在堂外同时并发拜佛求神，这就极大的利用了CPU资源。简而言之：只有第一批抢头香的在排队，之后大家都不必排队了，代码如下。

```
public class God {
    private volatile static God god;
    private God(){}
    public static God getInstance() {//庙是开放的不用排队进入
        if (god == null) {//如果头柱香未产生，这批抢香人进入堂内排队。
            synchronized(God.class){
                if (god == null) {//只有头香造了神，其他抢香的白排队了
                    god = new God();
                }
            }
        }
        //此处头柱香产生后不必再排队
        return god;
    }
}
```

其实在这之上还发展出了各种各样的单例模式变种，我们这里只讲了最基础的两种，其实他们都各有优缺，我们

化占用内存，加锁解锁更是一种浪费，还有同步效率低等问题，如果上帝不是很占空间那就没必要去懒汉延迟加载，越复杂问题越多，风险越大。

大道至简，无为而治。

本文作者：凸凹

设计模式(62)

分享：

精简阅读 生成封面

赞 5

上一篇：Java Excel、Word 转HTML

下一篇：SpringBoot自动配置原理

相关文章

设计模式六大原则（3）：依赖倒置原则

设计模式是什么鬼（命令模式）

设计模式是什么鬼（工厂方法）

23种设计模式介绍-外观模式

23种设计模式介绍-工厂模式

23种设计模式介绍-生成器模式

评论：

1 条评论，访客：1 条，站长：0 条

0% 好评

☆ ☆ ☆ ☆ ☆

好评：(0%)

中评：(0%)

差评：(0%)

傻不拉几的小傻瓜 发布于： 2018年7月23日 上午11:41

赞 0

博主用生动形象的比喻给我们介绍了痴汉和懒汉模式，非常不错

发表评论

评分 ☆ ☆ ☆ ☆ ☆

您的评论一针见血（必填，该内容可在后台设置）

昵称

邮箱

网址

发表评论