单例模式及其扩展在 Web 开发中的应用分析①

刘耀钦

(郧阳师范高等专科学校 计算机科学系, 十堰 442700)

摘 要: 单例模式是一种 Web 开发中广泛应用的设计模式, 应用该模式的类只能在类体内部实例化本类唯一的实例对象, 并通过该类的静态成员方法提供给整个应用程序, 起到全局共享的作用. 该种模式的应用可以降低应用程序的资源耗费、提高运行效率, 但当系统任务繁重或一个实例对象难以胜任时, 就显得捉襟见肘, 多例模式作为单例模式的扩展, 通过生成有限个实例对象, 既能避免类体本身实例对象的无序生成, 又很好的解决了单个实例对象所不能完成的任务, 可以说是单例模式优点的延续和扩展.

关键词: 模式; 单例; 多例; 变量

Application Analysis of Singleton Pattern and its Extension in Web Development

LIU Yao-Qin

(Department of Computer Science of Yunyang Teachers' College, Shiyan 442700, China)

Abstract: The singleton pattern is a design pattern used for Web development,the class instantiate the only object inside the class applied the singleton pattern, and provide the object to the whole application through the static methods of the class, play a shared global role. Application of this kind of mode can reduce the application's resource consumption and improve operational efficiency. But it is difficult when competency system task or an instance exists only. Multiple instances model is as an extension of singleton pattern. It can avoid disorderly generating the class body object through generating a finite object instance.It is a very good solution to finish the task unfinished by the only object also. This model can be said to be continuation and expansion of the singleton pattern.

Key words: patter; singleton; multiple instances; variables

1 引言

随着 Web 开发中模式设计的不断发展与进步, 基于单例模式及其扩展的应用已得到广泛使用, 诸如数据库连接池、线程池、缓存等都会用到单例模式, 就是说这些对象所属的类只能产生一个实例, 如果生成多个实例就会导致行为异常、资源耗费过量等一系列问题[1]. 特别是基于 B/S 的数据库应用程序开发会经常用到数据库连接池对象, 通常情况下, 我们会将数据库连接整合成一个类, 当我们需要的时候通过 new 运算符来产生一个连接对象, 然而当应用程序中相关数据表的操作比较多时, 就会需要产生很多数据库连接对象, 这样不仅重复生成了相同引用的对象, 而且造成了大量系统资源的浪费, 这也正是笔者在开发人力资源管理系统和学生报到注册系统时遇到的相当苦恼和困惑的问题. 单例模式及其扩展的应用很好的解决了这个问题, 该模式中, 数据库连接对象是唯一的, 并且被整个应用程序对象所共享且不能被复制, 既避免了 new 运算符的滥用又节约了系统资源, 从而提高了应用程序的执行效率. 多例模式作为单例模式的应用扩展, 通过生成有限个实例对象的方式, 既延续了单例模式的应用又扩展了单例模式的优点.

2 单例模式解析

模式就像是 OOP 开发人员的配方, 每种模式都提供了所需的成分, 我们可以自定义模式的元素来解决特定的编程问题[2]. 在 OOP 中, 创建的一个对象只负

① 收稿时间:2013-09-20;收到修改稿时间:2013-10-25

责某一项特定任务，任何时候，应用程序中只允许某个类的一个实例存在，这样既保证了数据操作的唯一性，又避免了过多资源的浪费，这就是单例模式的职责. 应用单例模式的单例类通常至少拥有被标记为 private 的构造函数、保存实例的 static 成员变量和访问该实例对象的 static 成员方法三个要素，通过 private 的构造函数自身实例化并且向当前类内部存储的实例返回一个引用，以供整个页面全局共享.

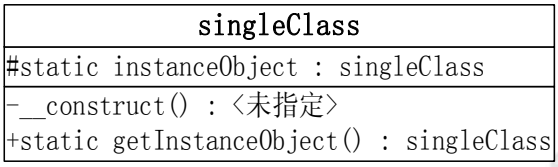

图 1 应用单例模式单例类基本构件

通常编写如下格式的类体作为单例模式的基本应用:

```
class singleClass{
    private static  $classObject = null; //声明静态私有型的成员变量，用于存储类的实例引用
    private function __construct(){ //将构造函数声明为私有型，避免在类体外用 new 实例化
        echo "i am here!";
    }
    public static function getSingleObject(){ //用于获取类的实例的静态成员方法，保证在首次调用该方法时实例化，以后不再实例化
        if(!isset(self::$classObject)) //判断成员变量是否设置
            self::$classObject = new self(); //如果没设置当前成员变量，则通过 new 实例化
        return self::$classObject; //返回类中存储类实例的引用
    }
    public function __clone(){ //重写__clone 方法，避免实例被复制
        echo "Object does not allow copied!";
    }
}
singleClass::getSingleObject(); //直接调用类的成员方法，具体是否 new 则方法体判断
```

在众多设计模式中，单例模式是一个比较简单的模式[5]，其主要特点是避免其他对象实例化自身或复制自身实例，从而确保类自身实例访问的唯一性，并且给 Web 页面提供了一个全局访问的接口. 单例模式提供了全局唯一的访问入口，易于控制可能发生的冲突，可以严格的控制客户访问它以及何时访问它[3]. 单例模式的应用不仅大大降低了程序模块、函数、类体之间的耦合度，而且提高了它们内部的聚合力.

在实际应用中，有些对象可能只需要一个，无谓的产生对象不仅会造成资源耗费，而且很有可能会引起程序混乱、命名冲突，更严重的会出现一些很难找到原因的程序 bug. 此种情况下，程序员通常会通过在程序的头部或配置文件里创建一个全局变量，然后把需要产生的对象值赋给它，最后在具体使用它的时候调用它. 可问题的关键就在于程序员难以判断何时使用它，如果该对象本身非常占用资源的话，该全局变量的声明无疑是非常失败的. 一种比较好的做法是在具体使用的时候再调用，即将该对象整合成一个类，并且在第一次调用时生成一个实例，以后其他任何情况调用均不再用 new 来生成新的实例.

从这个意义上来说，单例模式是全局变量的替代，但性能优于全局变量. 可是单例模式的本质并非是简单的替代全局变量，其实质是将数据封装于一个类中，且提供一个方法用于返回该类本身的唯一的实例对象，不允许在类体外被任意创建新的实例对象. 在使用全局变量时要坚持唯一性、不变性、共享性三个原则，唯一性是指在整个程序中有且仅能有一个，不能产生命名冲突，不变性是指其该全局变量的引用恒定不变，共享性则是指该引用值的变化必须对所有对象可见. 可实际开发中，系统功能模块繁多、程序结构复杂，想容易实现三个原则的有效统一有一定的困难，这也是 OOP 开发使用单例模式而尽可能回避全局变量的原因之一.

3 new运算开销

内存是暂时存储程序及数据的空间，是衡量计算机运算效率的一个重要指标. 内存的使操作包括申请内存、销毁内存以及修改内存大小等[4]. 内存的使用要非常注意防止内存泄露的情况发生，如果所申请的内存在应用程序使用完后没有及时释放，随着应用程序对象的不断生成，这些程序会极大的消耗系统直至耗

尽为止, 对任何平台来说都是一种可怕的情况.

在 PHP 中程序中, 声明的变量、类体、函数等均和内存的申请释放紧密相关. 在程序运行时, PHP 中所有的数据类型都要加载到内存中, 包括复合型的数据类型对象. 从逻辑上讲, PHP 内存管理由栈内存、堆内存、数据段、代码段等组成, 数据段通常存储初始化过的变量, 代码段是一块放在程序代码的内存区域, 栈空间存储占用空间较小的数据类型, 堆空间用于存储点用空间较大的数据类型. 结合上例, 在没有应用单例模式的情况下实例化 3 个对象, 它们在内存中的分配情况如图 2 所示:

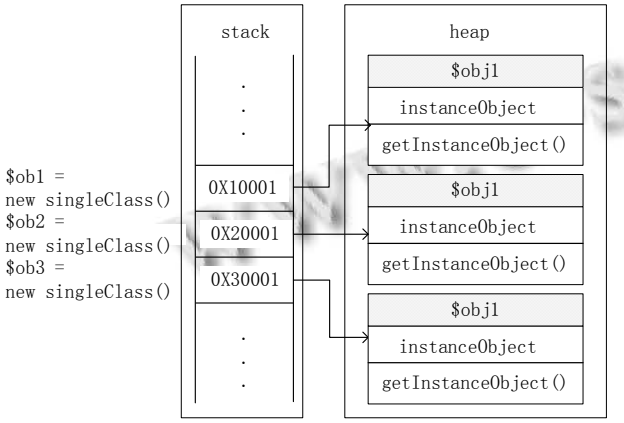

图 2　普通类创建实例在内存的分配

由图 2 可以看出, 普通类实例对象的引用存储于 stack, 而实例本身存储于 heap. 每一次 new 操作均会在内在中开辟一块区域用于存储实例对象, 所产生的实例对象在内在中相互独立, 互不干涉, 实际上, 每次 new 运算产生的实例均有相同的属性和方法, 无规律无限制的产生实例无疑会使增大系统内存开销, 造成资源的无端耗费. 在一台配置 2G 内存与 4×2.67GCPU 的计算机上, 实例化含有计算 1 千万次递增加 1 运算成员方法的类的一个对象, 需耗时约 2.5 秒. 由此可见, 对应用程序来说, 编写尽可能降低系统压力且使内存空间得到合理利用的程序结构有多么重要.

4　单例模式的扩展

单例模式最大的一个特点就是它只有一个实例, 程序中所有对象的任务分配均由这一个实例完成, 这个可以避免资源耗费的特点固然好, 但当对象的任务比较繁重或一个实例对象难以完成时, 该实例就会显得捉襟见肘, 在这种即要保持单例模式的优点又要回避其缺点的情况下, 只有改进单例模式的应用, 即生成有限数量 N 个实例对象, 就是这单例模式的扩展, 我们称之为有上限多例模式. 两种模式的共同之处在于它们的实例对象都不能被外界直接实例化, 只能在类体内部自身实例化, 同时通过一个静态成员方将该实例提供给外界, 不同之处是实例化的对象数量不同, 扩展后的单例类可以生成有限个实例对象, 通常应用于生成有限个连接、有限个用户的场景. 比如设计一个钟表类, 输出当时时间, 则只能生成该类的时、分、秒三个实例对象.

因为, 多例模式可以创建有限数量 N 个实例对象, 所以当这个上限值为 1 时, 多例模式就是一个普通的单例模式, 当 N 的值不多大时, 可以像单例模式一样, 使用 N 个静态成员变量存储 N 个实例对象, 这种处理办法自然不适用 N 值很大的情况. 另外, 究竟 N 的值有多大, 程序员事先无法估计, 所以在基于 PHP 的 Web 实际开发中, 使用一个 static 数组来存储 N 个实例对象.

动态时间可以通过有上限多例模式来实现. 时间有时(hour)、分(minute)、秒(second)三个属性组成, 分别用以表示类体的三个实例对象, 每个属性均有自己的取值范围, 我们声明一个应用多例模式的指针类, 用 static 数组分别存储时、分、秒三个实例对象的引用, 然后输出当前系统的时间. 在使用有上限多例模式的时候要注意的是上限的控制条件, 此例中实例对象的数量为 3, 所以, 类体本身在生成实例对象时要首先比对已生成实例对象的数量与指定实例对象的数量. 通常情况下, 在类的私有构造函数内部获取已生成实例对象的数量即数组中元素个数来实现两个数值的比对操作.

```
print "<meta http-equiv='refresh' content='1'>"; //第 1 秒刷新面面一次, 保证获取到最新时间
class pointer{
  private static $poType = array(); //static 类型的数组成员
  private static $s = "The Current Time is:";
  private function __construct($key){ //private 类型的构造函数, 不允许在类体外调用
    if(count(self::$poType)>2){ //比对已生成实例对象的数量
```

```
            die("The Count of Class pointer's Object is less Than 3!");
        }
    }
    public static function getPoType($key){ //static 成员方法, 用于生成类的实例
        if(!isset(self::$poType[$key])) //判断当前实例是否已生成
            self::$poType[$key] = new self($key);
        else
            die("The Current Object $key has been Created!");
        self::$poType[$key]->returnTime($key); //返回当前实例对象
    }
    public static function returnTime($key){ //将生成的 3 个实例分别赋以对应系统时间部分
        self::$s.= date($key).":";
        if(count(self::$poType)>2)
            echo substr_replace(self::$s,"",-1); //输出当前时间
        }
    }
    $h = pointer::getPoType("H"); //生成时针实例
    $m = pointer::getPoType("i"); //生成分针实例
    $s = pointer::getPoType("s"); //生成秒针实例
```

5 总结

Web 开发中, 单例模式的应用非常广泛. 单例模式的应用可以给应用程序带来诸如代资源耗费、弱耦合度、实例对象唯一等特点, 单例模式可以使应用程序在全局共享唯一实例对象的情况下得到高效率运行. 但当应用系统任务较繁重或一个实例对象难以胜任时, 就显得捉襟见肘, 多例模式作为单例模式的扩展, 很好的解决了这个问题, 多例模式通过在单例模式的基础上创建有限个实例对象的方式来分担完成系统任务, 即避免了实例对象的无序生成, 又完成了系统资源的有效分配任务.

参考文献

1 Freeman E, Freeman E, Sierra K, Bates B. Head First 设计模式.北京:中国电力出版社,2005:169–177.

2 McArthur K. PHP 高级程序设计模式、框架与测试. 北京:人民邮电出版社,2009:17–25.

3 夏浩波.单例模式的设计与应用.电脑开发与应用,2011,1:58–59.

4 深入理解 PHP 内核: Thinking In PHP Internals.http:// www.php-internals.com/book/?p=chapt06/06-00-memory-management. 2013-10.

5 秦小波.设计模式之禅.北京:机械工业出版社,2010:58–64.