私塾在线《研磨设计模式》
——跟着CC学设计系列精品教程

本节课程概览

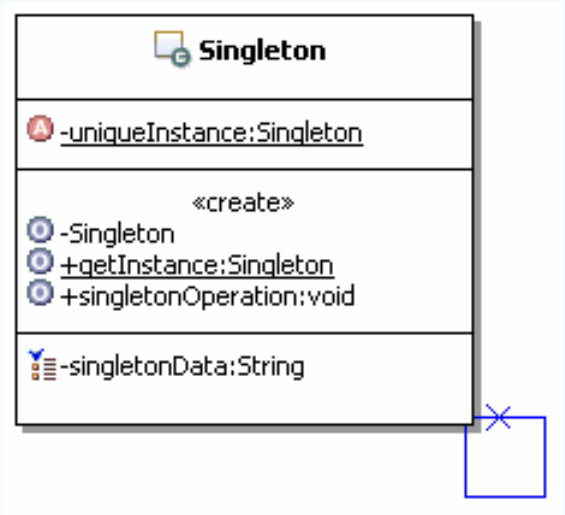
- 学习单例模式
  - 一：初识单例模式
    - 包括：定义、结构、参考实现
  - 二：体会单例模式
    - 包括：场景问题、不用模式的解决方案、使用模式的解决方案
  - 三：理解单例模式
    - 包括：认识单例模式、懒汉式和饿汉式实现、延迟加载的思想、缓存的思想、Java中缓存的基本实现、利用缓存来实现单例模式、单例模式的优缺点、在Java中一种更好的单例实现方式、单例和枚举
  - 四：思考单例模式
    - 包括：单例模式的本质、何时选用

初识单例模式

- 定义

  保证一个类仅有一个实例，并提供一个访问它的全局访问点。

- 结构和说明



Singleton：

负责创建Singleton类自己的唯一实例，并提供一个getInstance的方法，让外部来访问这个类的唯一实例。

体会单例模式

- 读取配置文件的内容

  现在要读取配置文件的内容，该如何实现呢？
- 不用模式的解决方案

  直接参看代码示例
- 存在的问题

  在系统运行期间，系统中会存在很多个AppConfig的实例对象，这会严重浪费系统资源。

  把上面的描述进一步抽象一下，问题就出来了：在一个系统运行期间，某个类只需要一个类实例就可以了，那么应该怎么实现呢？

- 使用模式的解决方案

  直接参看代码示例

理解单例模式

- 认识单例模式

1：单例模式的功能

单例模式的功能是用来保证这个类在运行期间只会被创建一个类实例，并提供一个全局唯一访问这个类实例的访问点。

2：单例模式的范围

是一个ClassLoader及其子ClassLoader的范围

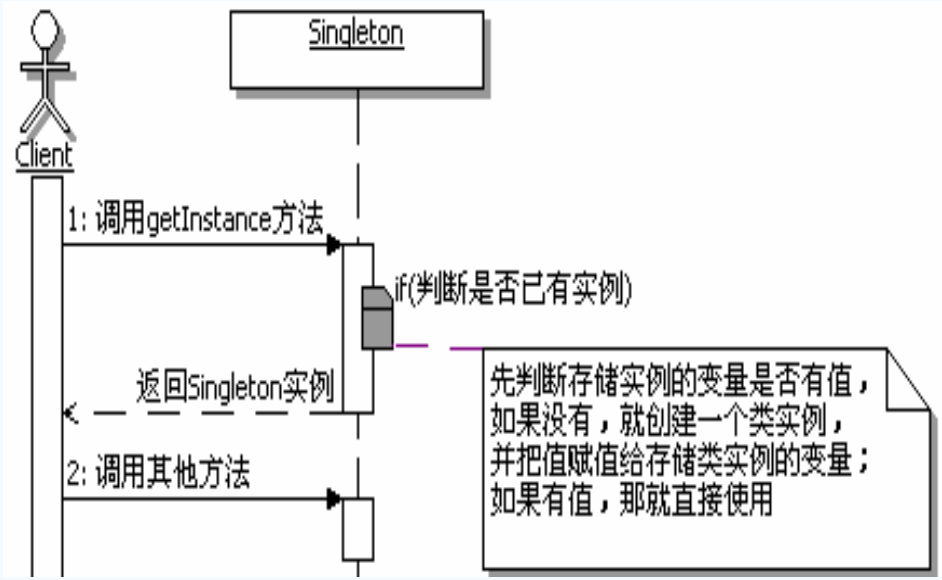
3：单例模式的命名

一般建议单例模式的方法命名为：getInstance() 。

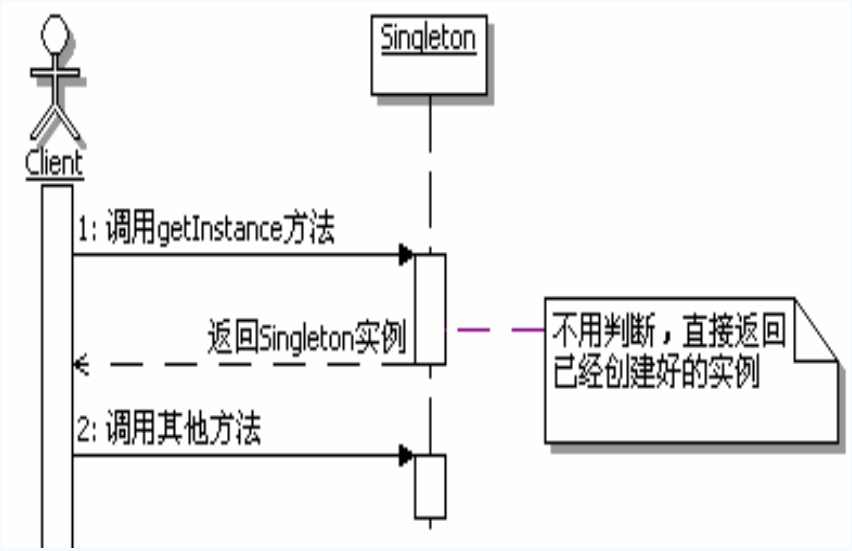
单例模式的名称：单例、单件、单体等等，翻译的不同，都是指的同一个模式

理解单例模式

- 懒汉式和饿汉式实现
  分步骤代码示例
- 单例模式的调用顺序示意图



懒汉式调用顺序示意图



饿汉式调用顺序示意图

理解单例模式

- 延迟加载的思想

  什么是延迟加载呢？

  通俗点说，就是一开始不要加载资源或者数据，一直等到马上就要使用这个资源或者数据了，躲不过去了才加载，所以也称Lazy Load，不是懒惰啊，是“延迟加载”，这在实际开发中是一种很常见的思想，尽可能的节约资源。

- 缓存的思想

  单例模式的懒汉式实现还体现了缓存的思想，缓存也是实际开发中非常常见的功能。

  简单讲就是，如果某些资源或者数据会被频繁的使用，可以把这些数据缓存到内存里面，每次操作的时候，先到内存里面找，看有没有这些数据，如果有，那么就直接使用，如果没有那么就获取它，并设置到缓存中，下一次访问的时候就可以直接从内存中获取了。从而节省大量的时间，当然，缓存是一种典型的空间换时间的方案。

理解单例模式

- Java中缓存的基本实现
  直接参看代码示例

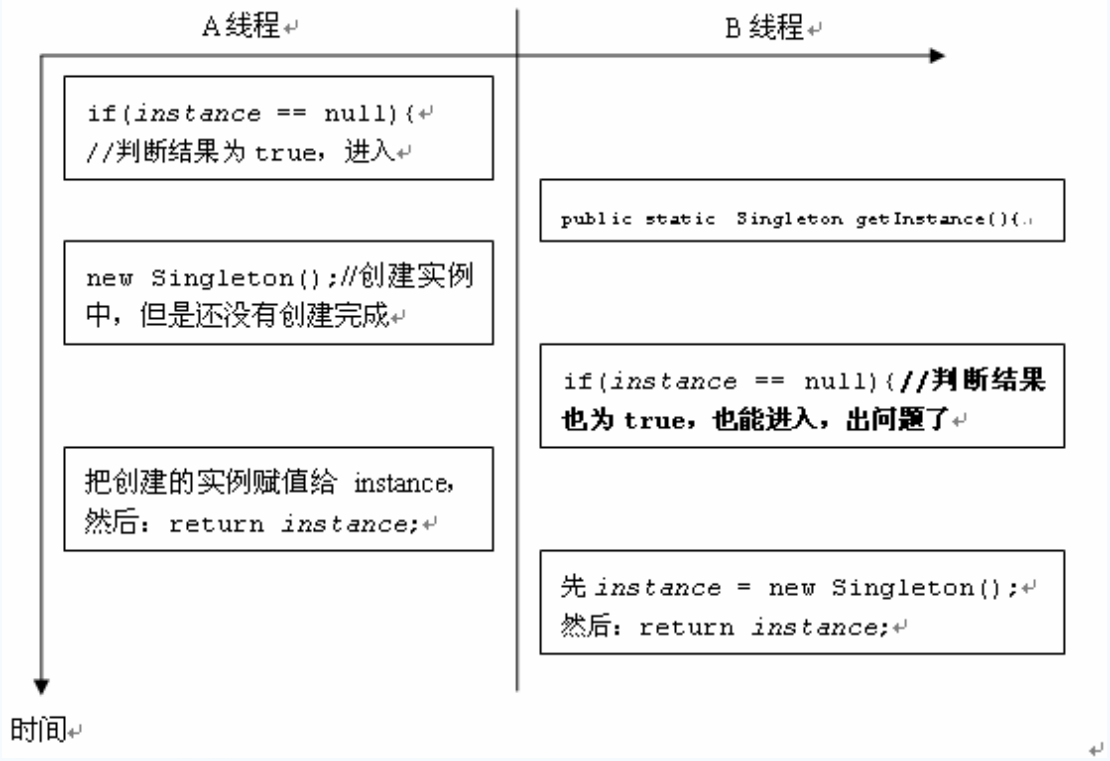
- 利用缓存来实现单例模式
  直接参看代码示例

理解单例模式

- 单例模式的优缺点

1：时间和空间：懒汉式是典型的时间换空间，饿汉式是典型的空间换时间

2：线程安全：（1）不加同步的懒汉式是线程不安全的



理解单例模式

（2）饿汉式是线程安全的，因为虚拟机保证了只会装载一次

（3）如何实现懒汉式的线程安全呢？

加上synchronized即可

（4）双重检查加锁

所谓双重检查加锁机制，指的是：并不是每次进入getInstance方法都需要同步，而是先不同步，进入方法过后，先检查实例是否存在，如果不存在才进入下面的同步块，这是第一重检查。进入同步块过后，再次检查实例是否存在，如果不存在，就在同步的情况下创建一个实例，这是第二重检查。这样一来，就只需要同步一次了，从而减少了多次在同步情况下进行判断所浪费的时间。

双重检查加锁机制的实现会使用一个关键字volatile，它的意思是：被volatile修饰的变量的值，将不会被本地线程缓存，所有对该变量的读写都是直接操作共享内存,从而确保多个线程能正确的处理该变量。

注意:在Java1.4及以前版本中，很多JVM对于volatile关键字的实现有问题，会导致双重检查加锁的失败，因此本机制只能用在Java5及以上的版本。

理解单例模式

- 在Java中一种更好的单例实现方式

  Lazy initialization holder class模式，这个模式综合使用了Java的类级内部类和多线程缺省同步锁的知识，很巧妙的同时实现了延迟加载和线程安全。

- 单例和枚举

  按照《高效Java 第二版》中的说法：单元素的枚举类型已经成为实现Singleton的最佳方法。

  为了理解这个观点，先来了解一点相关的枚举知识，这里只是强化和总结一下枚举的一些重要观点，更多基本的枚举的使用，请参看Java编程入门资料：

  （1）Java的枚举类型实质上是功能齐全的类，因此可以有自己的属性和方法

  （2）Java枚举类型的基本思想：通过公有的静态final域为每个枚举常量导出实例的类

  （3）从某个角度讲，枚举是单例的泛型化，本质上是单元素的枚举

  用枚举来实现单例非常简单，只需要编写一个包含单个元素的枚举类型即可

思考单例模式

- 单例模式的本质

  单例模式的本质是：**控制实例数目**

- 何时选用单例模式

  当需要控制一个类的实例只能有一个，而且客户只能从一个全局访问点访问它时，可以选用单例模式，这些功能恰好是单例模式要解决的问题