深蓝学院
shenlanxueyuan.com

单例九品

C++ 的单例实现讨论

主讲人： 李 伟

深蓝学院
shenlanxueyuan.com

目录

CONTENTS

学习前提：对 C++ 基础有一定的了解

通过学习，您将：

- 了解单例模式
- 对 C++ 语法的综合性应用
- 了解 static initialization order fiasco 与 C++ 标准流的初始化细节

单例介绍

- 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。（引用自《设计模式》）
- “公告栏” 式的功能

实现目标

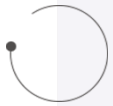
- 无需显式提供参数的初始化
- 如果不希望用户按照某种方式使用，那么应想方法明确禁止（尽量不要列于文档中）
- 如果希望用户按某种方式使用，那么应设计得尽量高效，至少对于大部分常用接口如此

对于将讨论的九次实现（改进）

- 每个实现都会在原有实现基础上进行优化，解决原有实现的问题
- 优化有的简单，有的复杂
- 每次讨论中都会包含问题，欢迎思考
- 并非每个实现都足够健壮可用
- 实现可能存在 Bug，欢迎讨论
- 可能存在更好的实现，欢迎讨论

第一品：

- 基本思路：使用面向对象的方式对要访问的数据进行封装
- 优点：符合基本的面向对象编程方法
  - 使用类对单例中的内容进行封装
  - 使用类的构造函数初始化数据成员，析构函数释放资源
- 缺点
  - 完全没有阻止用户构造该类的多个对象



第二品：

- 基本思路
  - 将构造、析构函数设置为私有成员
  - 使用静态成员函数获取唯一的实例
- 优点
  - 可以防止用户不小心构造新的实例
  - （C++11 开始）系统会保证函数内部的静态成员初始化是多线程安全的
- 特点
  - 缓式初始化（lazy initialization）
- 缺点
  - 没有完全阻止用户构造该类的多个对象

第三品（常见实现，可用）：

- 基本思路
  - 通过限制拷贝（移动）构造，拷贝（移动）赋值进一步杜绝非法复制
- 优点
  - 杜绝构造新的实例
- 缺点
  - 由于多线程安全所引入的额外成本

第四品：

- 基本思路
  - 引入全局对象，区分单例的初始化与调用
  - 静态函数用于初始化；全局对象用于后续调用
- 优点
  - 减少了调用所付出的成本
- 特点
  - 热初始化（eager initialization）
- 缺点
  - static initialization order fiasco 问题，程序存在较大隐患

第五品：

- 基本思路
  - 引入初始化类
  - 初始化类是单例类的子类，可以访问单例类的所有成员
  - 通过初始化类的实例作为纽带，一定程度上控制初始化顺序
- 优点
  - 初始化类可以精确控制初始化时机
- 缺点
  - 似乎可以解决 static initialization order fiasco 问题，但实际上程序包含了更深层次的隐患：可能出现未定义的行为。

第六品：

- 基本思路1——强制编译期初始化
  - constinit / consteval
- 基本思路2——运行期平凡缺省初始化
  - 使用基本数据类型代替抽象数据类型作为全局对象
  - 通过引用计数控制销毁时机
- 优点
  - 全局对象平凡初始化
  - 同时精确控制初始化与销毁
- 缺点
  - 多线程不安全

第七品：

- 基本思路
  - 为引用计数引入线程安全操作
- 优点
  - 全局对象初始化/销毁多线程安全
- 缺点
  - 指针有被修改的风险

第八品（可用）：

- 基本思路
  - 使用引用代替指针
    - 方法1：将原始指针封装到类内部，使用函数提供引用接口
    - 方法2：使用 placement new
- 优点
  - 避免指针赋值而产生的问题
- 缺点
  - 代码组织相对复杂，不易扩展

第九品（可用）：

- 基本思路
  - 对单例类的功能逻辑与单例逻辑进行划分，分别放入不同的部分
  - 使用CRTP模拟“基类-派生类”行为
- 优点
  - 更容易支持不同的单例实例
  - 使用模板不会引入运行期成本
- 缺点
  - 暂无

- 第三品与第九品的性能对比

- C++ 标准 IO 流的实现方式
  - ios_base::init 类
  - GCC 的实现示例

- 如果在实现 单例A 的逻辑时：
  - 部分实现逻辑在源文件而非头文件中
  - 源文件中的逻辑使用了 单例B 的内容
  - 那么一定要将 单例B 的头文件置于 单例A 的头文件中

深蓝学院
shenlanxueyuan.com

感谢聆听