Design Patterns

单例模式

单例模式是一种创建型设计模式，确保一个类只有一个实例，并提供全局访问点。单例模式分为懒汉模式和饿汉模式。

饿汉模式（Eager Initialization）

核心思想： 在程序启动时就创建单例实例，即类加载时就完成实例化。

特点：

- 提前申请内存资源，在类加载时就初始化静态实例
- 天然线程安全，无需额外的同步机制
- 实现简单，代码清晰
- 可能造成资源浪费，如果实例一直未被使用

适用场景：

- 单例对象占用资源较少
- 单例对象在程序运行期间必然会被使用
- 不需要延迟加载的场景

懒汉模式（Lazy Initialization）

核心思想： 延迟创建单例实例，只在第一次使用时才进行实例化。

特点：

- 延迟加载，按需创建，节约资源
- 存在线程安全问题，需要额外的同步机制（如双重检查锁定）
- 首次调用时有初始化开销，可能产生延迟
- 实现相对复杂，需要考虑多线程环境下的内存可见性和指令重排序问题

线程安全实现要点：

- 使用双重检查锁定（Double-Checked Locking）模式
- 使用原子变量（atomic）保证指针的线程安全访问
- 使用内存屏障（memory fence）防止指令重排序

- C++11 标准解决了内存屏障问题，提供了更可靠的实现方式

适用场景：

- 单例对象占用资源较大
- 单例对象可能不会被使用
- 需要延迟加载以提升程序启动速度的场景

选择建议

- **优先使用饿汉模式**：实现简单，线程安全，无性能损耗，除非有明确的延迟加载需求
- **谨慎使用懒汉模式**：只在确实需要延迟加载且资源占用显著的情况下使用
- **现代C++推荐**：使用局部静态变量实现懒汉模式，C++11保证了局部静态变量初始化的线程安全性（Meyers' Singleton）