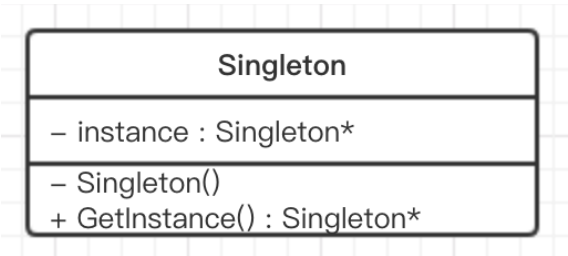
单例模式

资料 C++单例模式

01.简述

单例模式 是设计模式中最简单的形式之一，其目的是使得类的一个对象成为系统中的唯一实例。这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一中访问其唯一对象的方式，可以直接访问，不需要实例化该类的对象。

02.UML结构图



03.要点

- 单例类有且只有一个实例。（构造函数私有，杜绝外界创建）
- 单例类必须自行创建自己的唯一实例。（内部创建一个唯一实例，例如提供一个该类的静态私有成员）
- 单例类必须给所有其他对象提供这一实例。（提供一个静态共有函数，用于创建或获取其本省的静态私有对象，例如GetInstance()）
  除此之外，还有一些关键点需要注意：
- 线程安全（双检锁 - DCL，double - check - locking）
- 资源释放

04.几种实现

01.局部静态变量

这种方式很常见，实现简单，而且无需担心单例的销毁问题。

```cpp
//01.局部静态变量
class Singleton1
{
public:
    static Singleton1* GetInstance();

private:
    Singleton1() {}
};
Singleton1* Singleton1::GetInstance()
{
    static Singleton1 instance;
    return &instance;
}

```

02.懒汉式/饿汉式

懒汉式:

- 第一次调用时才进行初始化，避免浪费内存
- 必须要加锁来保证线程安全，但加锁会影响效率

饿汉式:

- 不需要加锁即可保证线程安全
- 类加载式就初始化，浪费内存

```cpp

//02.懒汉式 （非多线程安全，需要双检测来保证效率与线程安全性）
class Singleton2
{
public:
    static Singleton2* GetInstance();
    static void DestoryInstance();

private:
    Singleton2(){}

private:
    static Singleton2* m_pSingleton;
    //static mutex m_mutex; //锁
};
//懒汉式
Singleton2* Singleton2::m_pSingleton = nullptr;

Singleton2* Singleton2::GetInstance()
```

```
{
    if (m_pSingleton == nullptr){

        //std::lock_guard<mutex> lock(m_mutex); //会自动释放的锁

        if (m_pSingleton == nullptr){
            m_pSingleton = new Singleton2();
        }
    }

    return m_pSingleton;
}

void Singleton2::DestoryInstance()
{
    if (m_pSingleton)
    {
        delete m_pSingleton;
        m_pSingleton = nullptr;
    }
}


////////////////////////////////////////////////////////////////////////////////

//03.饿汉式 (线程安全)
class Singleton3
{
public:
    static Singleton3* GetInstance();
    static void DestoryInstance();

private:
    Singleton3(){}

private:
    static Singleton3* m_pSingleton;
};
//饿汉式
Singleton3* Singleton3::m_pSingleton = new Singleton3();

Singleton3* Singleton3::GetInstance()
{
    return m_pSingleton;
}

```

```
65 void Singleton3::DestoryInstance()
66 {
67     if (m_pSingleton)
68     {
69         delete m_pSingleton;
70         m_pSingleton = nullptr;
71     }
72 }
73
```