<https://blog.csdn.net/weixin_42762089/article/details/87969619> <https://cloud.tencent.com/developer/article/1504684> <https://blog.csdn.net/sxc1414749109/article/details/80151453> <https://blog.csdn.net/XindaBlack/article/details/106126108>

#pragma once

#include <iostream>

#include<list>

#include<string>

Class ItestExplam

{

Public:

Virtual const std::string& MyNameIs() const = 0;

Virtual const std::string& GetModel() const = 0;

Virtual const std::string& GetServerContainerName() const = 0;

Virtual ~ItestExplam(){}

};

Typedef std::list<ItestExplam\*> TestExplamList;

Class ItestMagaeSingLet

{

Public:

Virtual const std::string& MyNames() const = 0;

Virtual int Initialize() = 0;

Virtual ItestExplam\* QueryTestExplam(const char\* const strComponentName,const int iInstanced) = 0;

Virtual TestExplamList QueryTestExplam(const char\* const strComponentName) = 0;

Virtual TestExplamList GetQueryTestExplamList() const = 0;

Virtual void Shutdown() = 0;

Virtual ~ItestMagaeSingLet(){}

};

Typedef ItestMagaeSingLet\* (\*ITEST\_MAGAE\_SINGLET\_INSTANCE)();

Extern “C”

{

ItestMagaeSingLet\* GetOneInstance();

}

Class CtestMageSingLet:public ItestMagaeSingLet

{

Public:

CtestMageSingLet(const char\* const strMyName);

Virtual ~CtestMageSingLet();

Const std::string& MyNames() const;

ItestExplam\* QueryTestExplam(const char\* const strComponentName,const int iInstanced) ;

TestExplamList QueryTestExplam(const char\* const strComponentName) ;

TestExplamList GetQueryTestExplamList() const ;

Virtual void SetCommunicationProxy() = 0;

Virtual int Initialize() = 0;

Virtual void Shutdown() = 0;

Int LoadDeviceCom(const char\* const strGetComInstanceIfName);

Protected:

CtestMageSingLet();

Private:

Std::string m\_strMyName;

TestExplamList m\_listTextExamp;

};

//这里有几个问题，

//第一 extern “C”

//第二 关于单例模式的使用

//第三 关于宏与模板类的区别，还有常量的区别优缺点

//第四 const用法。Const char\* char\* const

//第五 static用法

//第六 命名规则

// 下面的方法是CtestMageSingLet的Initialize()纯虚函数

#define DEFINE\_EXAMPLE\_MANAGER(class\_exam\_manger)\

Private:\

Static ItestMagaeSingLet\* m\_pInstance;\

Public:\

Virtual ~class\_exam\_manger();\

Static ItestMagaeSingLet\* GetInstance();\

Public:\

Int Initialize(); \

Void SetLocalContainerName(); \

Void SetCommunicationProxy(); \

Void Shutdown(); \

\

Protected:\

Class\_exam\_manger():CtestMageSingLet(){}\

Class\_exam\_manger(const char\* const strMyName);\

Class Csingletion

{

Public:

~Csingletion();

Csingletion(const Csingletion&) = delete;

Csingletion& operator=(const Csingletion&) = delete;

Static Csingletion\* get\_instance();

Private:

Csingletion();

Public:

Void SetSomeValue(int iNest);

Int GetSomeValue(int iNest);

Private:

Std::list<int> m\_iTestList;

};

UML类图符号各种关系说明以及举例：

https://www.cnblogs.com/duanxz/archive/2012/06/13/2547801.html#:~:text=%20uml%E4%B8%AD%E7%94%A8%E7%A9%BA%E5%BF%83%E7%AE%AD%E5%A4%B4%E5%92%8C%E8%99%9A%E7%BA%BF%E8%A1%A8%E7%A4%BARealize%E5%85%B3%E7%B3%BB%EF%BC%8C%E7%AE%AD%E5%A4%B4%E6%8C%87%E5%90%91%E5%AE%9A%E4%B9%89%E7%BA%A6%E5%AE%9A%E7%9A%84%E5%85%83%E7%B4%A0%E3%80%82%20%E5%85%B3%E8%81%94%EF%BC%88Association%EF%BC%89%20%EF%BC%9A%E5%85%83%E7%B4%A0%E9%97%B4%E7%9A%84%E7%BB%93%E6%9E%84%E5%8C%96%E5%85%B3%E7%B3%BB%EF%BC%8C%E6%98%AF%E4%B8%80%E7%A7%8D%E5%BC%B1%E5%85%B3%E7%B3%BB%EF%BC%8C%E8%A2%AB%E5%85%B3%E8%81%94%E7%9A%84%E5%85%83%E7%B4%A0%E9%97%B4%E9%80%9A%E5%B8%B8%E5%8F%AF%E4%BB%A5%E8%A2%AB%E7%8B%AC%E7%AB%8B%E7%9A%84%E8%80%83%E8%99%91%E3%80%82%20uml%E4%B8%AD%E7%94%A8%E5%AE%9E%E7%BA%BF%E8%A1%A8%E7%A4%BAAssociation%E5%85%B3%E7%B3%BB%EF%BC%8C%E7%AE%AD%E5%A4%B4%E6%8C%87%E5%90%91%E8%A2%AB%E4%BE%9D%E8%B5%96%E5%85%83%E7%B4%A0%E3%80%82,%E8%81%9A%E5%90%88%EF%BC%88Aggregation%EF%BC%89%20%EF%BC%9A%E5%85%B3%E8%81%94%E5%85%B3%E7%B3%BB%E7%9A%84%E4%B8%80%E7%A7%8D%E7%89%B9%E4%BE%8B%EF%BC%8C%E8%A1%A8%E7%A4%BA%E9%83%A8%E5%88%86%E5%92%8C%E6%95%B4%E4%BD%93%EF%BC%88%E6%95%B4%E4%BD%93%20has%20a%20%E9%83%A8%E5%88%86%EF%BC%89%E7%9A%84%E5%85%B3%E7%B3%BB%E3%80%82

1. 依赖（Dependency）：A变化B也变，反之不成立，B依赖A

带箭头的虚线表示Dependency关系，箭头指向被依赖元素。

2. 泛化（Generalization）：继承（特殊个体 is kind of 一般个体）关系，

带空心三角箭头的实线线表示泛化关系，箭头指向一般个体（父类）。

3. 实现（Realize）：最常用于接口。A是接口类，B是实现类

空心三角箭头和虚线表示实现关系，箭头指向接口类。

4. 关联（Association）：元素间的结构化关系，是一种弱关系，

被关联的元素间通常可以被独立的考虑。

实线表示Association关系，箭头指向被依赖元素。

4.1 聚合（Aggregation）：关联关系的一种特例，

表示部分和整体（整体 has a 部分）的关系。

带空心菱形头的实线表示聚合关系，菱形头指向整体。

4.2 组合（Composition）：组合是聚合关系的变种，

表示元素间更强的组合关系。如果是组合关系，

如果整体被破坏则个体一定会被破坏，

而聚合的个体则可能是被多个整体所共享的，

不一定会随着某个整体的破坏而被破坏。

带实心菱形头的实线表示组合关系，菱形头指向整体。

详细与举例说明

1. 依赖： 依赖总是单向的，譬如人买车，就是单向#include,车是被指向的，

class Person

{

void buy(Car car)

{

}

}