4. 外观模式

目录

4.1. 模式动机

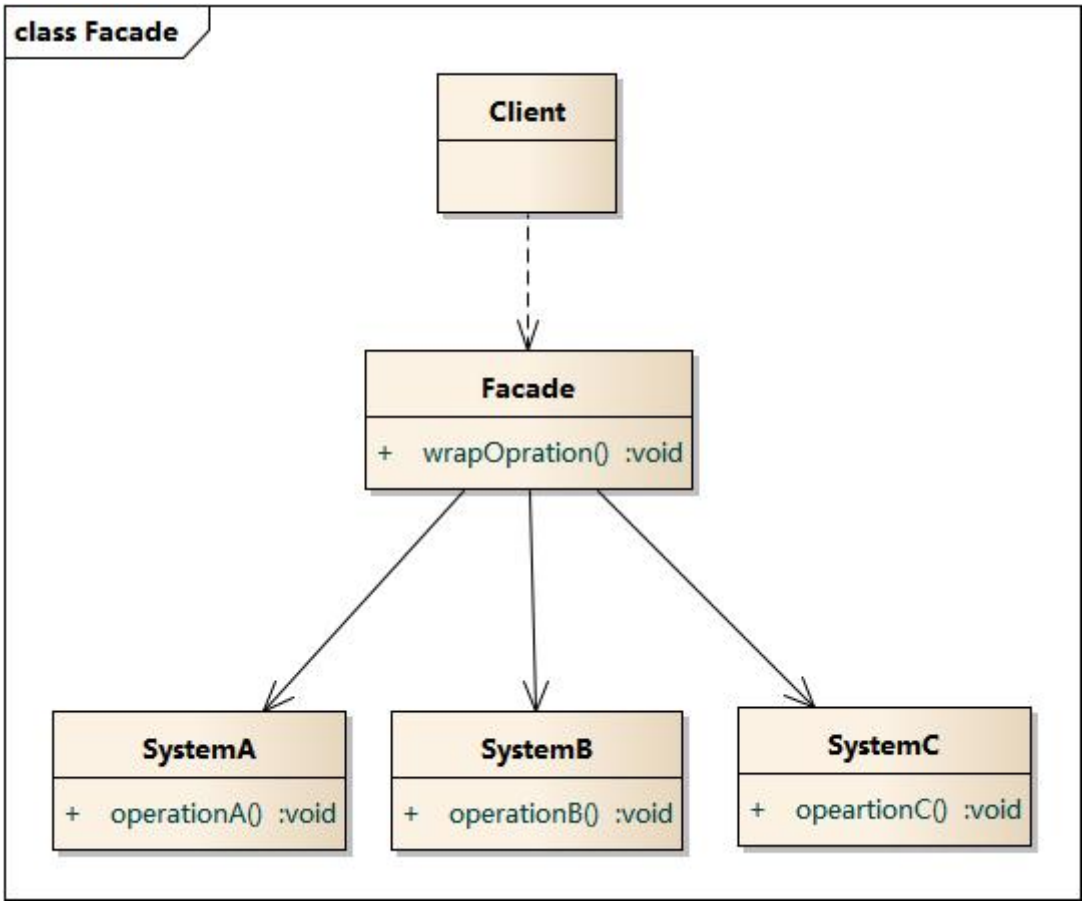
4.2. 模式定义

外观模式(Facade Pattern)：外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。外观模式又称为门面模式，它是一种对象结构型模式。

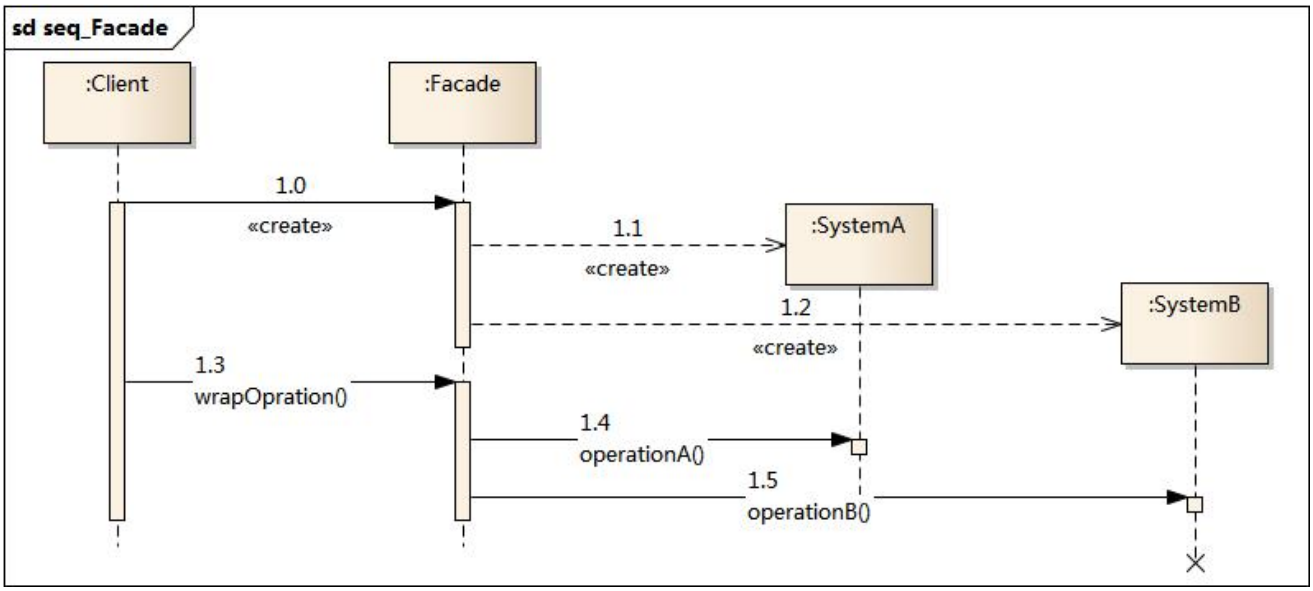
4.3. 模式结构

外观模式包含如下角色：

- Facade: 外观角色
- SubSystem:子系统角色



4.4. 时序图



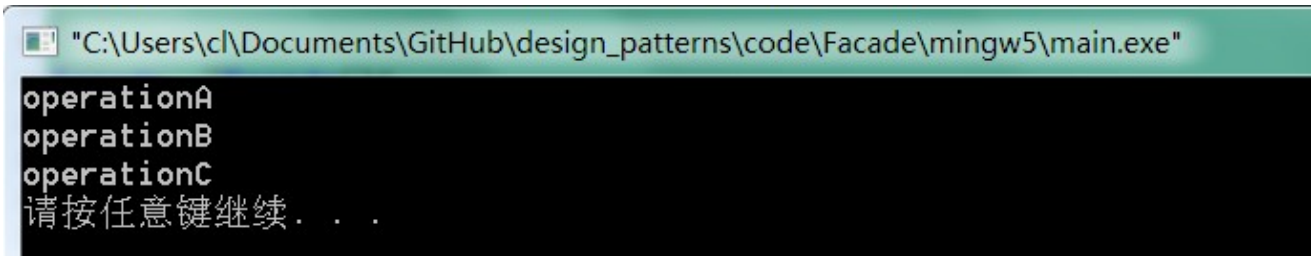
4.5. 代码分析

```cpp
#include <iostream>
#include "Facade.h"
using namespace std;

int main(int argc, char *argv[])
{
        Facade fa;
        fa.wrapOpration();

        return 0;
}
```

```cpp
///////////////////////////////////////////////////////////
//  Facade.h
//  Implementation of the Class Facade
//  Created on:      06-十月-2014 19:10:44
//  Original author: colin
///////////////////////////////////////////////////////////

#if !defined(EA_FD130A87_92A9_4168_9B33_7A925C47AFD5__INCLUDED_)
#define EA_FD130A87_92A9_4168_9B33_7A925C47AFD5__INCLUDED_

#include "SystemC.h"
#include "SystemA.h"
#include "SystemB.h"

class Facade
{

public:
        Facade();
        virtual ~Facade();

        void wrapOpration();

private:
        SystemC *m_SystemC;
        SystemA *m_SystemA;
        SystemB *m_SystemB;
};
#endif // !defined(EA_FD130A87_92A9_4168_9B33_7A925C47AFD5__INCLUDED_)
```

```cpp
////////////////////////////////////////////////////////
//  Facade.cpp
//  Implementation of the Class Facade
//  Created on:      06-十月-2014 19:10:44
//  Original author: colin
////////////////////////////////////////////////////////

#include "Facade.h"


Facade::Facade(){
        m_SystemA  = new SystemA();
        m_SystemB = new SystemB();
        m_SystemC = new SystemC();
}



Facade::~Facade(){
        delete m_SystemA;
        delete m_SystemB;
        delete m_SystemC;
}

void Facade::wrapOpration(){
        m_SystemA->operationA();
        m_SystemB->operationB();
        m_SystemC->opeartionC();
}
```

运行结果：



4.6. 模式分析

根据“单一职责原则”，在软件中将一个系统划分为若干个子系统有利于降低整个系统的复杂性，一个常见的设计目标是使子系统间的通信和相互依赖关系达到最小，而达到该目标的途径之一就是引入一个外观对象，它为子系统的访问提供了一个简单而单一的入口。 -外观模式也是“迪米特法则”的体现，通过引入一个新的外观类可以降低原有系统的复杂度，同时降低客户类与子系统类的耦合度。 - 外观模式要求一个子系统的外部与其内部的通信通过一个统一的外观对象进行，外观类将客户端与子系统的内部复杂性分隔开，使得客户端只需要与外观对象打交道，而不需要与子系统内部的很多对象打交道。 -外观模式的目的在于降低系统的复杂程度。 -外观模式从很大程度上提高了客户端使用的便捷性，使得客户端无须关心子系统的工作细节，通过外观角色即可调用相关功能。

4.7. 实例

4.8. 优点

外观模式的优点

- 对客户屏蔽子系统组件，减少了客户处理的对象数目并使得子系统使用起来更加容易。通过引入外观模式，客户代码将变得很简单，与之关联的对象也很少。
- 实现了子系统与客户之间的松耦合关系，这使得子系统的组件变化不会影响到调用它的客户类，只需要调整外观类即可。
- 降低了大型软件系统中的编译依赖性，并简化了系统在不同平台之间的移植过程，因为编译一个子系统一般不需要编译所有其他的子系统。一个子系统的修改对其他子系统没有任何影响，而且子系统内部变化也不会影响到外观对象。
- 只是提供了一个访问子系统的统一入口，并不影响用户直接使用子系统类。

4.9. 缺点

外观模式的缺点

- 不能很好地限制客户使用子系统类，如果对客户访问子系统类做太多的限制则减少了可变性和灵活性。
- 在不引入抽象外观类的情况下，增加新的子系统可能需要修改外观类或客户端的源代码，违背了“开闭原则”。

4.10. 适用环境

在以下情况下可以使用外观模式：

- 当要为一个复杂子系统提供一个简单接口时可以使用外观模式。该接口可以满足大多数用户的需求，而且用户也可以越过外观类直接访问子系统。
- 客户程序与多个子系统之间存在很大的依赖性。引入外观类将子系统与客户以及其他子系统解耦，可以提高子系统的独立性和可移植性。
- 在层次化结构中，可以使用外观模式定义系统中每一层的入口，层与层之间不直接产生联系，而通过外观类建立联系，降低层之间的耦合度。

4.11. 模式应用

4.12. 模式扩展

一个系统有多个外观类
    在外观模式中，通常只需要一个外观类，并且此外观类只有一个实例，换言之它是一个单例类。在很多情况下为了节约系统资源，一般将外观类设计为单例类。当然这并不意味着

在整个系统里只能有一个外观类，在一个系统中可以设计多个外观类，每个外观类都负责和一些特定的子系统交互，向用户提供相应的业务功能。

不要试图通过外观类为子系统增加新行为

不要通过继承一个外观类在子系统中加入新的行为，这种做法是错误的。外观模式的用意是为子系统提供一个集中化和简化的沟通渠道，而不是向子系统加入新的行为，新的行为的增加应该通过修改原有子系统类或增加新的子系统类来实现，不能通过外观类来实现。

外观模式与迪米特法则

外观模式创造出一个外观对象，将客户端所涉及的属于一个子系统的协作伙伴的数量减到最少，使得客户端与子系统内部的对象的相互作用被外观对象所取代。外观类充当了客户类与子系统类之间的“第三者”，降低了客户类与子系统类之间的耦合度，外观模式就是实现代码重构以便达到“迪米特法则”要求的一个强有力的武器。

抽象外观类的引入

外观模式最大的缺点在于违背了“开闭原则”，当增加新的子系统或者移除子系统时需要修改外观类，可以通过引入抽象外观类在一定程度上解决该问题，客户端针对抽象外观类进行编程。对于新的业务需求，不修改原有外观类，而对应增加一个新的具体外观类，由新的具体外观类来关联新的子系统对象，同时通过修改配置文件来达到不修改源代码并更换外观类的目的。

4.13. 总结

- 在外观模式中，外部与一个子系统的通信必须通过一个统一的外观对象进行，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。外观模式又称为门面模式，它是一种对象结构型模式。
- 外观模式包含两个角色：外观角色是在客户端直接调用的角色，在外观角色中可以知道相关的(一个或者多个)子系统的功能和责任，它将所有从客户端发来的请求委派到相应的子系统去，传递给相应的子系统对象处理；在软件系统中可以同时有一个或者多个子系统角色，每一个子系统可以不是一个单独的类，而是一个类的集合，它实现子系统的功能。
- 外观模式要求一个子系统的外部与其内部的通信通过一个统一的外观对象进行，外观类将客户端与子系统的内部复杂性分隔开，使得客户端只需要与外观对象打交道，而不需要与子系统内部的很多对象打交道。
- 外观模式主要优点在于对客户屏蔽子系统组件，减少了客户处理的对象数目并使得子系统使用起来更加容易，它实现了子系统与客户之间的松耦合关系，并降低了大型软件系统中的编译依赖性，简化了系统在不同平台之间的移植过程；其缺点在于不能很好地限制客户使用子系统类，而且在不引入抽象外观类的情况下，增加新的子系统可能需要修改外观类或客户端的源代码，违背了“开闭原则”。
- 外观模式适用情况包括：要为一个复杂子系统提供一个简单接口；客户程序与多个子系统之间存在很大的依赖性；在层次化结构中，需要定义系统中每一层的入口，使得层与层之间不直接产生联系。