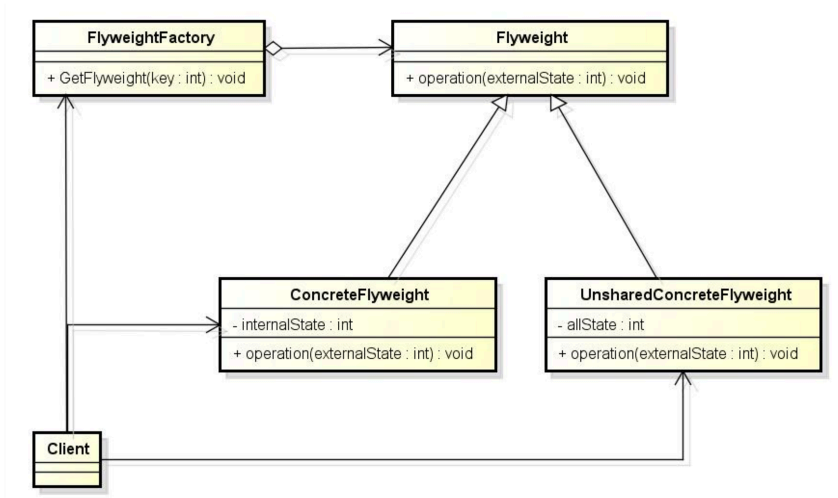
动机:在软件系统采用纯粹对象方案的问题在于大量细粒度的对象会很快充斥在系统中,从而带来很高的运行时代价. 主要指内存需求方面的代价. 如何在避免大量细粒度对象问题的同时,让外部客户程序仍然能够透明的使用面向对象的方式来进行操作.

定义:运用共享技术有效地支持大量细粒度的对象. <设计模式>GoF

要点总结:面象对句很好地解决了抽象性的问题,但是作为一个运行在机器中的程序实体,我们需要考虑对象的代价问题. Flyweight主要解决面向对象的代价问题. 一般不触及面向对象的抽象性问题. 享元模式采用对象共享的做法来降低系统中对象的个数,从而降低细粒度对象给系统带来的内存压力. 在具体实现方面, 要注意对象状态的处理. 对象的数量太大从而导致对象内存开销加大. 什么样的数量才算大? 需要我们仔细根据具体应用情况进行评估,而不能凭空臆断.



FlyweightFactory
+ GetFlyweight(key : int) : void
Flyweight
+ operation(externalState : int) : void
ConcreteFlyweight
- internalState : int
+ operation(externalState : int) : void
UnsharedConcreteFlyweight
- allState : int
+ operation(externalState : int) : void
Client

Flyweight：描述一个接口，通过这个接口flyweight可以接受并作用于外部状态；

ConcreteFlyweight：实现Flyweight接口，并为定义了一些内部状态，ConcreteFlyweight对象必须是可共享的；同时，它所存储的状态必须是内部的；即，它必须独立于ConcreteFlyweight对象的场景；

UnsharedConcreteFlyweight：并非所有的Flyweight子类都需要被共享。Flyweight接口使共享成为可能，但它并不强制共享。

FlyweightFactory：创建并管理flyweight对象。它需要确保合理地共享flyweight；当用户请求一个flyweight时，FlyweightFactory对象提供一个已创建的实例，如果请求的实例不存在的情况下，就新创建一个实例；

Client：维持一个对flyweight的引用；同时，它需要计算或存储flyweight的外部状态。

当要将一个对象进行共享时，就需要考虑到对象的状态问题了；不同的客户端获得共享的对象之后，可能会修改共享对象的某些状态；大家都修改了共享对象的状态，那么就会出现对象状态的紊乱。对于享元模式，在实现时一定要考虑到共享对象的状态问题。那么享元模式是如何实现的呢?

在享元模式中，有两个非常重要的概念：内部状态和外部状态。

内部状态存储于flyweight中，它包含了独立于flyweight场景的信息，这些信息使得flyweight可以被共享。而外部状态取决于flyweight场景，并根据场景而变化，因此不可共享。用户对象负责在必要的时候将外部状态传递给flyweight。

flyweight执行时所需的状态必定是内部的或外部的。内部状态存储于ConcreteFlyweight对象之中；而外部对象则由Client对象存储或计算。当用户调用flyweight对象的操作时，将该状态传递给它。同时，用户不应该直接对ConcreteFlyweight类进行实例化，而只能从FlyweightFactory对象得到ConcreteFlyweight对

象，这可以保证对它们适当地进行共享；由于共享一个实例，所以在创建这个实例时，就可以考虑使用单例模式来进行实现。

享元模式的工厂类维护了一个实例列表，这个列表中保存了所有的共享实例；当用户从享元模式的工厂类请求共享对象时，首先查询这个实例表，如果不存在对应实例，则创建一个；如果存在，则直接返回对应的实例。

享元模式:顾名思义就是羽量级模式或者蝇级模式，形容体量小的应用，该模式主要的设计目的是为了迎合系统大量相似数据的应用而生，减少用于创建和操作相似的细碎对象所花费的成本。大量的对象会消耗高内存，享元模式给出了一个解决方案，即通过共享对象来减少内存负载。

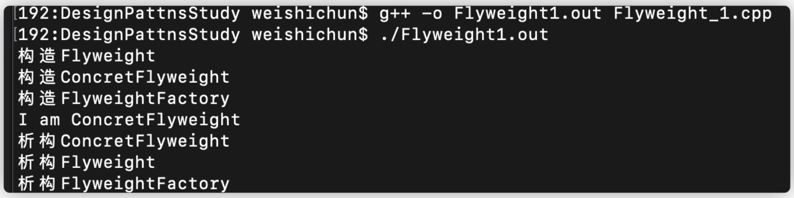
作用:通过复用相同的对象来减少对象的创建数量，创建更小的对象组，并通过共享实现重用。通过归类，将对象的属性分为内蕴状态和外蕴状态。要创建具体的享元对象，我们需要创建一个享元工厂来统一管理对象的生成和输出，享元工厂是实现享元模式的关键。

举个例子，享元模式可以看成是一个工具箱，而享元对象就是工具箱内的具体的工具，我们在使用工具的时候，不必每回临时的制造工具，而是直接从工具箱里找到工具进行使用，这样就大大节约了制造工具的成本时间和工具占用的空间。

享元模式比较迷惑在于理解两种状态的分类，内部状态是对象本身的属性，在生成对象以后一般不会进行改变，比如工具中的属性：名字、大小、重量等，还有就是我们一般需要一个关键性的属性作为其区别于其他对象的key，如工具的话我们可以把名称作为找到工具的唯一标识。

外部状态是对象的外部描述，是每个对象的可变部分，比如对工具的使用地点、使用时间、使用人、工作内容的描述，这些属性不属于对象本身，而是根据每回使用情况进行变化的，这就需要制作成接口进行外部调用，而外蕴状态的维护是由调用者维护的，对象内不进行维护。

```cpp
#include <iostream>
#include <vector>
using namespace std;
class Flyweight{
public:
    Flyweight(){ cout << "构造Flyweight" << endl;}
    virtual ~Flyweight(){ cout << "析构Flyweight" << endl;}
    virtual void Operator() = 0;
};
class ConcretFlyweight : public Flyweight{
public:
    ConcretFlyweight(){ cout << "构造ConcretFlyweight" << endl;}
    ~ConcretFlyweight(){ cout << "析构ConcretFlyweight" << endl;}
    void Operator(){
        cout << "I am ConcretFlyweight" << endl;
    }
};
class FlyweightFactory{
private:
    vector<Flyweight*> m_vecpWeights;
public:
    FlyweightFactory(){
        Flyweight *tmp =new ConcretFlyweight();
        m_vecpWeights.push_back(tmp);
        cout << "构造FlyweightFactory" << endl;
    }
    ~FlyweightFactory(){
        Flyweight *tmp = m_vecpWeights.at(0);
        delete tmp;
        tmp = NULL;
        cout << "析构FlyweightFactory" << endl;
    }
    Flyweight* GetFlyweight(int key){
        return m_vecpWeights.at(key);
    }
};
int main(){
    FlyweightFactory* pFactory = new FlyweightFactory();
    Flyweight * pFly = pFactory->GetFlyweight(0);
    pFly->Operator();
    delete pFactory;//一定要释放内存
    delete pFly;
    return 0;
}
```

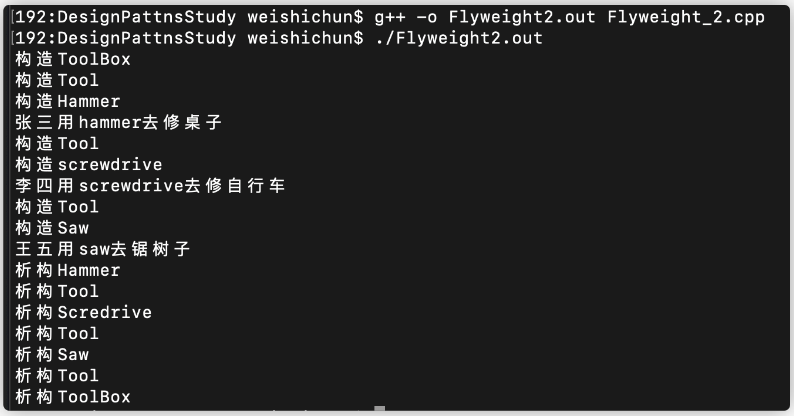


```
192:DesignPattnsStudy weishichun$ g++ -o Flyweight1.out Flyweight_1.cpp
192:DesignPattnsStudy weishichun$ ./Flyweight1.out
构造Flyweight
构造ConcretFlyweight
构造FlyweightFactory
I am ConcretFlyweight
析构ConcretFlyweight
析构Flyweight
析构FlyweightFactory
```

```cpp
#include <iostream>
#include <map>
#include <string>
#include <utility>
using namespace std;
class Tool
{ //=Flyweight
public:
    //内部状态
    string m_strName;
public:
    virtual void Use(string strPerson, string strWork) = 0; //外部状态
    Tool() { cout << "构造Tool" << endl; }
    virtual ~Tool() { cout << "析构Tool" << endl; }
};
class Hammer : public Tool
{ //=ConcretFlyWeight
public:
    Hammer()
    {
        m_strName = "hammer";
        cout << "构造Hammer" << endl;
    }
    ~Hammer() { cout << "析构Hammer" << endl; }
    void Use(string strPersion, string strWork)
    {
        cout << strPersion << "用" << m_strName << "去" << strWork << endl;
    }
};
class Screwdrive : public Tool
{ //=ConcretFlyWeight
```

```cpp
public:
    Screwdrive()
    {
        m_strName = "screwdrive";
        cout << "构造screwdrive" << endl;
    }
    ~Screwdrive() { cout << "析构Scredrive " << endl; }
    void Use(string strPersion, string strWork)
    {
        cout << strPersion << "用" << m_strName << "去" << strWork << endl;
    }
};
class Saw : public Tool
{ //=ConcretFlyWeight
public:
    Saw()
    {
        m_strName = "saw";
        cout << "构造Saw" << endl;
    }
    ~Saw() { cout << "析构Saw" << endl; }
    void Use(string strPersion, string strWork)
    {
        cout << strPersion << "用" << m_strName << "去" << strWork << endl;
    }
};
class ToolBox
{ //=FlyweightFactory
private:
    map<string, Tool *> m_mapTools;

public:
    ToolBox() { cout << "构造ToolBox" << endl; }
    ~ToolBox() {
        cout << "析构ToolBox" << endl;
    }
    Tool *GetTool(string strName)
    {
        map<string, Tool *>::iterator it = m_mapTools.find(strName);
        if (it != m_mapTools.end())
        {
            return (Tool *)it->second;
        }
        else
        {
            Tool *pTemp;
            if (strName == "hammer")
                pTemp = new Hammer();
            else if (strName == "screwdrive")
                pTemp = new Screwdrive();
            else if (strName == "saw")
                pTemp = new Saw();
            else
            {
                cout << "工具箱中没有:" << strName << endl;
            }
            if (NULL != pTemp)
            {
                m_mapTools.insert(pair<string, Tool *>(strName, pTemp));
            }
            return pTemp;
        }
    }
};
int main()
{
    ToolBox objToolBox;
    Tool *pToolA = objToolBox.GetTool("hammer");
    pToolA->Use("张三", "修桌子");
    Tool *pToolB = objToolBox.GetTool("screwdrive");
    pToolB->Use("李四", "修自行车");
    Tool *pToolC = objToolBox.GetTool("saw");
    pToolC->Use("王五", "锯树子");
    delete pToolA;
    delete pToolB;
    delete pToolC;
    return 0;
}
```

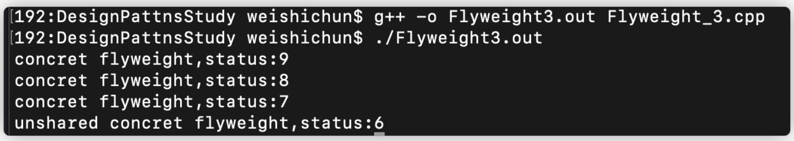


```
192:DesignPattnsStudy weishichun$ g++ -o Flyweight2.out Flyweight_2.cpp
192:DesignPattnsStudy weishichun$ ./Flyweight2.out
构造ToolBox
构造Tool
构造Hammer
张三用hammer去修桌子
构造Tool
构造screwdrive
李四用screwdrive去修自行车
构造Tool
构造Saw
王五用saw去锯树子
析构Hammer
析构Tool
析构Scredrive
析构Tool
析构Saw
析构Tool
析构ToolBox
```

```cpp
#include<iostream>
#include<map>
#include<string>
using namespace std;
class Flyweight{
public:
    virtual ~Flyweight(){};
    virtual void Operator(int outStatus) = 0;
};
class ConcretFlyweight : public Flyweight{
public:
    void Operator(int outStatus){
        cout << "concret flyweight,status:" << outStatus << endl;
    }
};
class UnsharedConcretFlyweight : public Flyweight{
public:
    void Operator(int outStatus){
        cout << "unshared concret flyweight,status:" << outStatus << endl;
    }
};
class FlyweightFactor{
private:
    map<string, Flyweight*> m_mapFlyweights;
public:
    Flyweight* GetFlyweight(string key){
        if(NULL == m_mapFlyweights[key]){
            m_mapFlyweights[key] = new ConcretFlyweight();
        }
        return m_mapFlyweights[key];
    }
};
int main(){
    int outStatus = 10;
    FlyweightFactor *pFactory = new FlyweightFactor();
    Flyweight *pFly1 = pFactory->GetFlyweight("X");
    pFly1->Operator(--outStatus);

    Flyweight *pFly2 = pFactory->GetFlyweight("Y");
    pFly2->Operator(--outStatus);

    Flyweight *pFly3 = pFactory->GetFlyweight("Z");
    pFly3->Operator(--outStatus);
    delete pFly1;
    delete pFly2;
    delete pFly3;
    delete pFactory;
    UnsharedConcretFlyweight* uf = new UnsharedConcretFlyweight();
    uf->Operator(--outStatus);
    delete uf;
    return 0;
}
```



```
192:DesignPattnsStudy weishichun$ g++ -o Flyweight3.out Flyweight_3.cpp
192:DesignPattnsStudy weishichun$ ./Flyweight3.out
concret flyweight,status:9
concret flyweight,status:8
concret flyweight,status:7
unshared concret flyweight,status:6
```