软件构造实验报告五

实验名称：享元模式编程实现

实验时间：2019.5.8

学号：E21614061

姓名：徐奕

所在院系：计算机科学与技术学院

所在专业：软件工程

【实验目的和要求】

a）熟悉并理解享元模式的原理与方法

b）熟练掌握享元模式的代码与方法

【实验原理】

动机

- 假设成立一个外包公司，主要给一些私营业主建网站。
- 商家客户都类似，要求也就是信息发布、产品展示、博客留言、论坛等功能。
- 各个客户要求差别不大，但客户数量多。

内部状态和外部状态

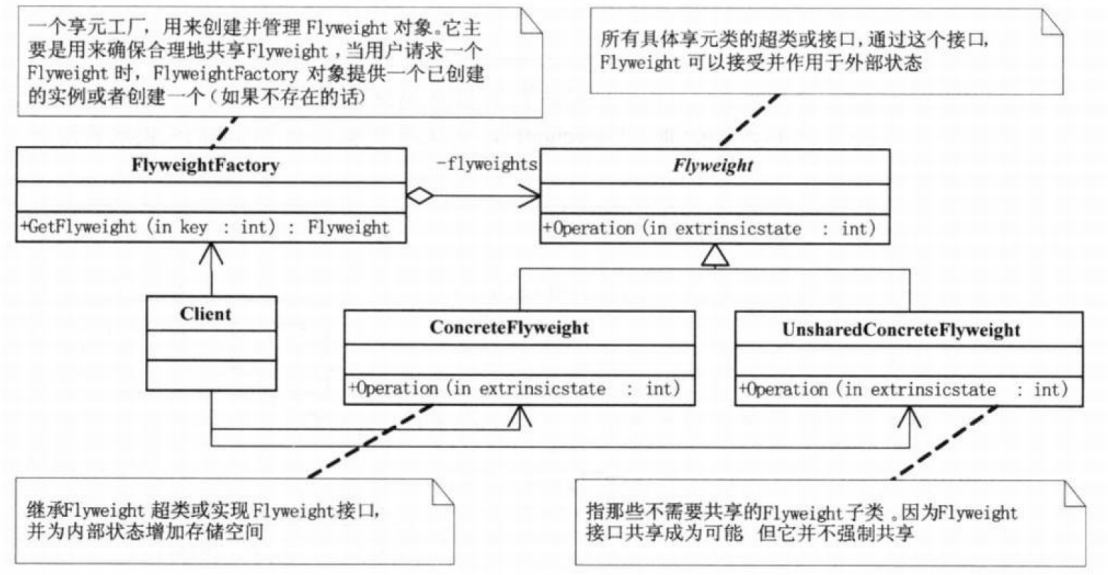
- 在享元对象内部并且不会随环境改变而改变的共享部分，可以成为是享元对象的内部状态。
- 随环境改变而改变的、不可以共享的状态就是外部状态。
- 享元模式可以避免大量非常相似类的开销。在程序设计中，有时需要生成大量细粒度的类实例来表示数据，如果能发现这些实例除了几个参数外基本都是相同的，有时就能大幅度减少需要实例化的类的数量。
- 如果能把那些参数移到类实例外面，在方法调用时将它们传递进来，就可以通过共享大幅度地减少单个实例的数目。

适用性

- 当以下情况成立时使用享元模式：

- 一个应用程序使用大量的对象，并且完全是由于使用大量的对象，造成很大的存储开销。
- 对象的大多数状态都可变为外部状态，并且如果删除对象的外部状态，那么可以用相对较少的共享对象取代很多组对象。
- 应用程序不依赖于对象表示。由于 Flyweight 对象可以被共享，对于概念上明显有别的对象，表示测试将返回真值。

结构



参与者

- 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，并为内部状态增加空间。ConcreteFlyweight 对象必须是可共享的。它所存储状态必须是内部的；即，它必须独立于 ConcreteFlyweight 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。
  - 确保合理地共享 flyweight。当用户请求一个 flyweight 时，Flyweight 对象提供一个已创建的实例或者创建一个（如果不存在的话）。
- Client
  - 维持一个对 flyweight 的引用。
  - 计算或存储一个（多个）flyweight 的外部状态。

协作

- Flyweight 执行时所需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。

- 用户不直接对 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。

【实验内容】

假设成立一个外包公司，主要给一些私营业主建网站。商家客户要求的功能包括信息发布、产品展示、博客留言等功能。

共有 a-j 10 个客户，其中 a-c 客户需要信息发布，d-f 需要产品展示，g-j 客户需要博客功能。

利用享元模式模拟该公司建立网站，输出如下：

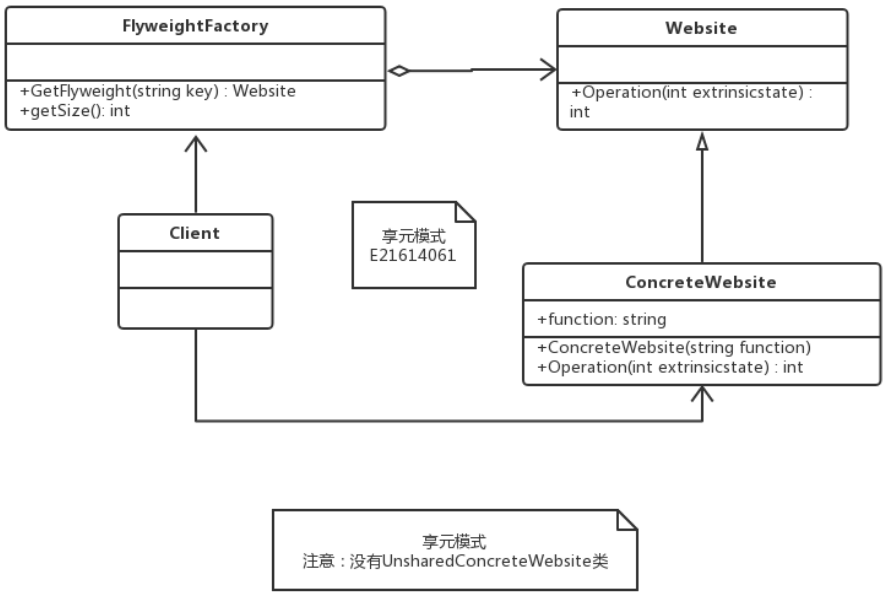
客户 a: 信息发布

客户 b：信息发布

。。。

网站分类总数为：3

【实验 UML 图】



【实验代码与函数】

在本次实验中：客户的号码就是外部状态，应该由专门的对象来处理，在这里简化为 char 类型的 id。
另外注意没有 UnsharedConcreteFlyweight 类

Website 类即 Flyweight 类
ConcreteWebsite 类即 ConcreteFlyweight 类

```
#include<iostream>
#include<map>
#include<vector>
using namespace std;

/*
假设成立一个外包公司，主要给一些私营业主建网站。商家客户要求的功能包括信息发布、产品展示、博客留言等功能。
共有a-j 10 个客户，其中a-c 客户需要信息发布，d-f 需要产品展示，g-j 客户需要博客功能。
利用享元模式模拟该公司建立网站，输出如下：

客户a：信息发布
客户b：信息发布
。。。
```

```
网站分类总数为：3
 */


class Website{
public:
    virtual void Operation(char extrinsicstate) = 0;
};

class ConcreteWebsite: public Website {
public:
    ConcreteWebsite(string function){
        this->function = function;
    }
    void Operation(char extrinsicstate){
        cout<<"客户"<<extrinsicstate<<": "<<function<<endl;
    }
private:
    string function;
};


class FlyweightFactory{
private:
    map<string, ConcreteWebsite*> flyweights;
public:
    FlyweightFactory(){
        flyweights.insert(make_pair("信息发布", new ConcreteWebsite("信
息发布")));
        flyweights.insert(make_pair("产品展示", new ConcreteWebsite("产
品展示")));
        flyweights.insert(make_pair("博客功能", new ConcreteWebsite("博
客功能")));
    }
    Website* GetFlyweight(string key){
        if(! flyweights.count(key)){
            flyweights.insert(make_pair(key, new
ConcreteWebsite(key)));
            cout<<"添加状态: "<<key<<"成功"<<endl;
        }
        return (Website*) flyweights[key];
    }
    int getSize(){
        return flyweights.size();
```

```
    }
};

int main(){
    FlyweightFactory *factory = new FlyweightFactory();
    vector<Website*> pool;
    for(char i = 'a'; i <= 'j'; i++){
        Website *f;
        if(i >= 'a' && i <= 'c'){
            f = factory->GetFlyweight("信息发布");
            f->Operation(i);
        }else if(i >= 'd' && i <= 'f'){
            f = factory->GetFlyweight("产品展示");
            f->Operation(i);
        }else{
            f = factory->GetFlyweight("博客功能");
            f->Operation(i);
        }
        pool.push_back(f);
    }
    cout<<"网站分类总数为: "<<factory->getSize()<<endl;
    return 0;
}
```

【实验结果】

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

XYs-MacBook-Pro:享元模式 reacubeth$ cd "/Users/reacubet
/reacubeth/Desktop/享元模式/"flyweight
客户a: 信息发布
客户b: 信息发布
客户c: 信息发布
客户d: 产品展示
客户e: 产品展示
客户f: 产品展示
客户g: 博客功能
客户h: 博客功能
客户i: 博客功能
客户j: 博客功能
网站分类总数为: 3
XYs-MacBook-Pro:享元模式 reacubeth$
```

【实验总结】

①本次实验掌握了享元模式的原理与方法，在选择 flyweight 时，要注意选择合适的享元类。

②按照实际情况分情况讨论具体的 Flyweight 类。

③不管建几个网站，只要是‘产品展示’，都是一样的，只要是‘博客’，也是完全相同的，但这样是有问题的，给企业建的网站不是一家企业，它们的数据不会相同，所以至少它们都应该有不同的账号（id 值）。实际上这样写没有体现对象间的不同，只体现了它们共享的部分。