

[首页](#) [问答](#) [专栏](#) [讲堂](#) [标签](#) [发现](#) [Q](#)[立即登录](#)[免费注册](#)

【设计模式】组合模式之神经网络应用

赞 | 1

收藏 | 0

[原](#) [深度学习](#) [神经网络](#) [组合模式](#) [设计模式](#) [C++](#)

254 次浏览

[Ender](#) 2017年11月06日发布

问题引入

试想，一个对象本身和由对象组成的一个集合都需要支持逻辑上相同的操作，实际实现可能不一样。既然在语义这个更高级抽象可以把两者统一，那么如果这两者都继承同一个基类岂不是更好？一个可以类比的例子是，目录本身可能包含多个目录，在没有子目录的情况下，目录的遍历是遍历该目录下的文件，反之，则需要递归的遍历子目录了，而对于扫描的目录的客户端来说(调用此API的程序员)实际上不需要关注和区分这两种情况，那么这就是一个抽象的机会，统一两者操作。这里给出组合模式的本质：

So, what is the Composite pattern about? Essentially, we try to give single objects and groups of objects an identical interface

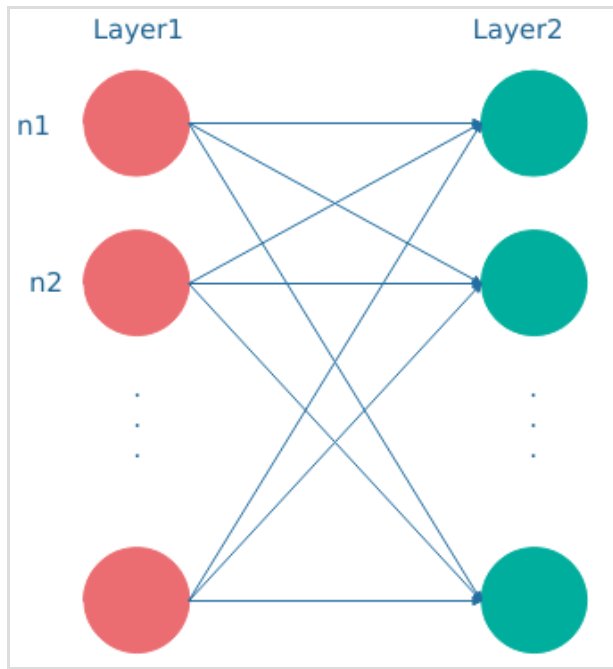
原理

显而易见的处理方式是，定义一个抽象基类 `componet`，并定义原始类和组合类公共操作(一般是遍历操作比如 `tranverse()`)，然后原始类和组合类各自继承即可，这种方式暂且不提。

这里提供一个另外个应用更加广泛和通用的组合模式实现方式，以神经网络基本神经元的连接作为例子。

神经网络基础

深度学习当前是以什么网络为基础的，所谓神经网络可以理解为由很多层，每层由很多个神经元，且各层神经元之间有特定关系的拓扑，简单的二层神经网络如下：



可以看到这个网络由橙色和绿色两层网络组成，每一层有若干个神经元，层和层之间的神经元存在连接关系，那么在这里，神经元本身就是 **原始类**，而神经网络层则是 **组合类** (若干神经元组成). 那么这两种类有什么公共的操作呢？

- 以单个神经元来看，一个神经元要和其他神经元建立连接关系 --> `build_connection`
- 以单个神经网络层来看，不同的层之间也需要建立连接关系 --> `build_connection`
- 单个神经元和层神经元建立连接. 排列组合一下，这种组合的个数是 $2*2=4$

而这个组合个数实际上就是类的成员函数的个数, 比如两个类A和B，分别代表神经元和层，需要支持两量互相连接操作需要如下几组成员函数：

```
A  a1,a2;
B  b1,b2;
a1.build_connection(a2);
a1.build_connection(b1);
b1.build_connection(b2);
b1.build_connection(a1);
```

推而广之，如果在来个平面层(由若干个单层神经网络组成)，那么个数就是 $3*3$. 随着类的增加，程序的重载成员函数规模失控，这违背了设计模式原则 **扩展开放，对修改关闭** .

怎么做呢？不同神经元(或者层)之间的连接，本质上可以抽象成一个遍历操作:对于单个神经元来说，遍历的结果就是只拿一个元素. 因此可以把抽象操作提升/抽象至基类，其他类继承并提供遍历迭代器接口即可.

实现

先给出神经元和神经网络层的类定义:

```

// 神经元
struct Neuron
{
    std::vector<Neuron *> in, out;
    unsigned int id;
    Neuron(){
        static int id = 1;
        this->id = id++;
    }
    // ...
};
// 单层神经元
struct NeuronLayer
{
    std::vector<Neuron> neus;
    NeuronLayer(int count) {
        while(count-- > 0) {
            neus.push_back(Neuron());
        }
    }
    // ...
};

```

然后想一下我们要抽象的操作，obj1.connect_to(obj2), 其中 obj1 和 obj2 可能是 NeuronLayer 或者 Neuron 之一，connect_to 实际上要分别遍历两个对象取到每一个神经元而建立连接. 抽象类如下:

```

template <typename Self>
template <typename T>
void SomeNeurons<Self>::connect_to(T& other) {
    for(Neuron &from : *static_cast<Self*>(this)){
        // 直接用*this本身会产生编译错误，因为抽象模板类并没有实现begin()和end()，而遍历需要它们, 这里转
        //for(Neuron &from : (*this)){
        for(Neuron & to : other) {
            from.out.push_back(&to);
            to.in.push_back(&from);
        }
    }
}

```

可以看到这个抽象类是一个带模板成员函数的模板类，connect_to 分别遍历了 this 类对象和指向的目标类对象获取元素，因为两个对象不一定一样，因此传入的模板类型也是不一样的. 接下来要干的活就剩下两个了

- 神经元和神经层继承抽象类，即可以获取 connect_to 能力
- 基类使用了 Range-based for loop，因此需要保证各子类都提供 begin 和 end 成员函数，因为Range语句的约束条件如下:

any expression that represents a suitable sequence (either an array or an object for which begin and end member functions or free functions are defined, see below) or a braced-init-list.

因此，按照上述描述新增/修改位置如下：

```
struct Neuron : public SomeNeurons<Neuron>
struct NeuronLayer : public SomeNeurons<NeuronLayer>
// 类Neuron，各自新增的成员函数，因为神经元不需要遍历，end直接取下一个元素。
Neuron *begin() {return this;}
Neuron *end()   {return this+1;}
// 类NeuronLayer，本来就是要遍历vector<Neuron>，直接用vector的begin和end包装即可。
std::vector<Neuron>::iterator begin() {return neus.begin();}
std::vector<Neuron>::iterator end()  {return neus.end();}
```

好了，主体工作基本完成，看一下当前类图结构如下：



总结

相较于比较基本的组合设计模式的实现方式(基类提供抽象接口，各子类分别实现)，这个抽象成都更高，连实现也放到抽象基类当中，而各子类提供的是遍历方式，解耦更加干净彻底；
更加有借鉴意义的是，这个实现方式可以完美的处理神经网络不同元素之间的关系，调用非常舒畅：

```
// point to layer
Neuron n6;
NeuronLayer l1(5);
n6.connect_to(l1);
std::cout<<n6<<std::endl;
```

参考

<https://leanpub.com/design-pa...>

<http://en.cppreference.com/w/...>

2017年11月06日发布 ...

赞 | 1

收藏 | 0

你可能感兴趣的文章

学习笔记DL002:AI、机器学习、表示学习、深度学习，第一次大衰退 362 浏览

深度学习浅析，以及又拍云图片鉴别的实践进阶 1 收藏，438 浏览

前端每周清单第 51 期: React Context API 与模式变迁, Webpack 与 Web 优化, AI 界面生成 8 收藏，580 浏览

评论

默认排序 时间排序



文明社会，理性评论

发表评论



Ender

562 声望

关注作者

发布于专栏


ender_614574

1 人关注

关注专栏

系列文章

【设计模式】空对象设计模式学习 197 浏览

产品	资源	商务	关于	关注	条款
热门问答	每周精选	人才服务	关于我们	产品技术日志	服务条款
热门专栏	用户排行榜	企业培训	加入我们	社区运营日志	内容许可
热门讲堂	徽章	活动策划	联系我们	市场运营日志	
最新活动	帮助中心	广告投放		团队日志	
技术圈	声望与权限	区块链解决方案		社区访谈	
找工作	社区服务中心	合作联系			
移动客户端	开发手册				扫一扫下载 App

Copyright © 2011-2018 SegmentFault. 当前呈现版本 17.06.16
浙ICP备 15005796号-2 浙公网安备 33010602002000号 杭州堆栈科技有限公司版权所有

CDN 存储服务由 又拍云 赞助提供