И. О. ШАЛЬНЕВ
Санкт-Петербургский Федеральный исследовательский центр Российской академии наук, Санкт-Петербург

ОСОБЕННОСТИ РАБОТЫ С КОМПЛЕМЕНТАРНЫМИ ОБЪЕКТАМИ В РАСПРЕДЕЛЕННОЙ ВИРТУАЛЬНОЙ СРЕДЕ

В статье рассматриваются особенности проектирования и использования комплементарных объектов. Приведенные примеры иллюстрируют парадигмы программирования, необходимые для правильного использования комплементарных объектов.

Введение. Динамическая реконфигурация информационно-вычислительных узлов сети является насущным вызовом в области программирования. Для решения проблемы динамической реконфигурации была разработана концепция распределенной виртуальной машины [1]. В основе распределенной виртуальной машины лежит объектно-ориентированное программирование (ООП) [2]. Так как каноничный объект ООП не является сетевой сущностью, то было предложено расширить эту парадигму определением комплементарного объекта. Разделенный на функциональную и программно-интерфейсную составляющие объект будем называть комплементарным. Части комплементарного объекта можно расположить на разных узлах сети, тем самым обеспечивая удаленное управление функциональным объектом. Управление осуществляется путем использования объекта программно-интерфейсной стороны. Именование программно-интерфейсного объекта соответствует именованию функционального с добавлением префикса V (от слова virtual). Несмотря на то, что использование обычных объектов ООП схоже с использованием объектов программно-интерфейсной стороны, они все же имеют различия, которые следует учитывать, а именно: обработка результата метода объекта, создание и деструкция комплементарных объектов.

Обработка результата метода объекта. Одной из особенностей в использовании комплементарного объекта является обработка результата вызванного метода. Если обычный объект считает результат на том же вычислительном устройстве, что и обрабатывает его, то у комплементарного объекта вычисление результата и его обработка могут выполняться на совершенно разных устройствах. В силу этого, ожидание результата выполнения метода в момент его вызова, как это осуществляется с методами классических ООП объектов (рис. 1*а*), неэффективно из-за возможных простоев процессора. Методы программного интерфейса комплементарных объектов не возвращают свои результаты средствами языка C++ и обозначены ключевым словом void. Обработка пришедшего результата должна осуществляется асинхронно посредством событийно-ориентированной методологии [3]. В момент, когда результат поступает на сторону программного интерфейса, виртуальная машина определяет объект, метод которого был вызван и подает сигнал о подготовленном результате. Такой сигнал обрабатывается функцией обратного вызова (callback) или как её по-другому называют – слотом. На рис. 1*б* показан пример обработки результата удаленного метода. В качестве примера был создан демонстрационный класс, метод которого удаленно складывает два числа. Здесь, третьим аргументом в метод передается обработчик результата, который в данном случае был определен как лямбда-функция, выводящая результат на экран.

```
Adder adder;
int res = adder.add(10, 23);
qInfo() << "result:" << res;
```

а)

```
auto adder = new VAdder( getProcessor() );
adder->add(10, 23, [](int res){
    qInfo() << "result:" << res; //result=33
});
```

б)

Рис. 1. Способы обработки результата удаленного метода

Создание и деструкция комплементарных объектов. Как создание, так и деструкция объекта программного интерфейса влечет за собой создание/деструкцию функционального объекта. Информация о необходимости создания/деструкции функционального объекта отправляется со стороны программного интерфейса в виде байт-кода [4]. Из-за того, что комплементарные объекты имеют асинхронную природу, то создание объектов в стеке может быть губительно. Так как по выходу из области видимости объект удалится, и в случае вызова удаленного метода

результат не успеет отправиться и, как следствие, обработаться. В данном случае его необходимо создать динамически и асинхронно удалить после возникновения какого-либо события, например, по возврату результата от функциональной стороны (рис. 2).

Несмотря на это, методы объектов могут иметь исключительно управляющий характер. Как правило, подобные методы ничего не возвращают, либо возвращают код ошибки. Такие объекты могут быть созданы в стеке. На функциональной стороне это будет выглядеть следующим образом: создание объекта, вызов метода, удаление объекта. Также, объекты, которые являются полем другого класса, могут быть определены в стеке, так как объект, агрегирующий виртуальный объект, управляет его жизненным циклом.

В случае, когда разработчик виртуального класса убежден, что создаваемый им класс не является управляющим и его методы требуют возврата результата, то он может обезопасить пользователя от возможных ошибок, скрыв конструкторы private, либо protected спецификатором доступа и определить статические методы создающие (create) и копирующие (copy) виртуальные объекты. Эти методы возвращают указатель на созданный или скопированный объект.

Существует подмножество классов, в основании которых лежат другие объекты. Создадим еще один демонстрационный класс Multiplier и соответствующий ему VMultiplier (рис. 2), единственная задача которого удаленно умножать два числа. Совершать это действие он будет путем итеративного сложения числа с самим собой при помощи уже созданного класса Adder. В конструктор передается указатель на созданный объект класса Adder, а по деструкции класса Multiplier он удаляется.

Реализация VMultiplier должна быть максимально абстрагирована от функционала и должна дублировать лишь сигнатуры методов, конструкторов, но никак не их логику. Поэтому процесс деструкции объектов adder и multiplier, показанных на рис. 2, будет следующим: сперва удалится интерфейсный объект multiplier, сформировав байт-код на удаление соответствующего функционального объекта. Затем, во время удаления функционального объекта multiplier, удалится агрегированный функциональный объект adder. Определив это, виртуальная среда функционального объекта сформирует байт-код на удаление интерфейсного объекта класса VAdder.

```
auto adder = VAdder::create( getProcessor() );
auto multiplier = VMultiplier::create( adder, getProcessor() );
multiplier->multiply(3, 4, [multiplier](int res)
{
    qInfo() << "Multiply:" << res;
    delete multiplier
});
```

Рис. 2. Создание вложенного виртуального объекта

Показанная выше имплементация удаленного произведения не единственная из возможных реализаций. В данном случае логика использования функционального объекта Adder реализована на функциональной стороне объекта Multiplier. Объект VMultiplier является программным интерфейсом удаленного доступа и не несет никакой алгоритмической составляющей. Можно создать альтернативную реализацию объекта Multiplier, путем переноса алгоритма умножения на сторону программного интерфейса [4]. Это будет обычный объект, метод которого будет последовательно запрашивать сумму до тех пор, пока не получит произведение.

Реализованный на рис. 3 метод будет вызывать удаленный метод сложения, при этом число вызовов будет определяться значением второго аргумента, в то время как объект класса VMultiplier вызовет удаленный метод всего лишь один раз. В данном случае это может показаться невыгодным, но существует ряд задач, которые не в состоянии выполниться на удаленном устройстве и тогда выполнение работы на месте может являться хорошим компромиссом.

```
void VMultiplierVA::_multiply(const int a, const int b)
{
    if(nullptr == mAdder){
        qWarning() « "mAdder is nullptr";
        emit multiplyReceived(mLastMultiply);
        return;
    }
    if(0 == b)
        emit multiplyReceived(mLastMultiply);
    else {
        auto tmp = (a>0)^(b>0)? 0-std::abs(a): std::abs(a);
        mAdder->add(mLastMultiply, tmp, [this, a, b](int res)
        {
            mLastMultiply = res;
            _multiply(a, b>0? b-1: b+1);
        });
    }
}
```

Рис. 3. Реализация метода вычисляющего произведение на стороне программного интерфейса

Заключение. По итогу работы была создана довольно простая и удобная реализация использования комплементарных объектов, основанная на современных методологиях асинхронного сетевого программирования. Подобная технология позволяет создавать сетевые приложения, с возможностью быстрого переконфигурирования с целью соответствия миру активно изменяющихся задач.

Работа выполнена в рамках реализации Государственного задания на 2020 г. № 0073-2019-0005.

ЛИТЕРАТУРА

1. Шальнев И.О. Подход к построению распределенной виртуальной машины на основе объектно-ориентированного программирования. Известия ТулГУ «в печати»
2. Буч Г., Максимчук Р.А., Энгл М.У., Янг Б. Дж., Коналлен Д., Хьюстон К.А. Объектно-ориентированный анализ и проектирование с примерами приложений: 3-е издание. М.: Вильямс, 2010. 720 с.
3. Шлее М. Qt 5.10. Профессиональное программирование на C++. СПб:БХВ-Петербург, 2018. 1072 с.
4. Шальнев И.О. Подход к построению распределенных систем на основе балансировки объема исполняемого кода между сетевыми узлами. Технологическая перспектива: новые рынки и точки экономического роста. Материалы конференции, 2018.

I.O. Shalnev (St. Petersburg Federal Research Center of the Russian Academy of Sciences, St. Petersburg)

Complementary Object Usage Features in Distributed Virtual Environment

The paper is concerned with complementary objects features, design and usage. The given examples illustrate programming paradigms are required in complementary object usage.