РЕАЛИЗАЦИЯ МУЛЬТИМЕТОДОВ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++

Д.Г. Шопырин
Научный руководитель – д.т.н., профессор А.А. Шалыто

Язык программирования C++ не предоставляет стандартных механизмов реализации мультиметодов. Однако известны разнообразные способы эмуляции этой возможности. Данная работа содержит краткое описание одного из таких подходов.

Введение

Мультиметод является общим случаем виртуального метода, или, другими словами, обычный виртуальный метод можно назвать унарным мультиметодом. Вызов виртуального метода зависит от конкретного типа объекта. Вызов же мультиметода зависит от конкретного типа нескольких учавствующих в вызове объектов.

Мультиметоды имеют большое прикладное значение. Классическим примером является задача пересечения геометрических фигур. Требуется реализовать функцию, проверяющую, пересекаются ли две геометрические фигуры (в общем случае, две или более фигур). Параметрами данной функции являются два указателя на базовый тип геометрической фигуры (например, `Shape`). Другими словами, конкретные типы геометрических фигур (*окружность*, *треугольник* и т.д.) неизвестны на момент компиляции программы. Пример использования подобной функции приведен ниже.

```
Shape* one = new Circle(/*parameters*/);
Shape* two = new Triangle(/*parameters*/);
bool intersects = CheckIntersection(one, two);
```

Существует два основных подхода к решению данной проблемы. Первый состоит в разработке максимально обобщенной функции пересечения геометрических фигур, что требует написания алгоритмически очень сложного кода.

Другой подход состоит в написании множества *простых* функций, пересекающих заранее известные типы геометрических фигур, например, прямоугольник и треугольник, прямоугольник и окружность и т.д. Очевидно, что написание таких *точечных* функций не сопряжено с решением сложных алгоритмических задач. Проблема состоит в механизме *диспетчеризации* вызовов этих *точечных* функций, или *специализаций*. Мультиметоды как раз и предоставляют необходимый для решения подобных задач механизм диспетчеризации.

Некоторые языки программирования, например Dylan [1], поддерживают мультиметоды напрямую. Однако наиболее распространенные *промышленные* объектно-ориентированные языки программирования, такие как C++, Java и C#, не поддерживают мультиметоды.

Существует два основных подхода к обеспечению поддержки мультиметодов:

- подход, основанный на *нестандартном* расширении компилятора (используется, например, в работе [3]);
- подход, основанный только на стандартных средствах используемого языка программирования (используется, например, в работе [2]).

Ниже приводится краткое описание способа обеспечения поддержки мультиметодов в языке C++ с использованием только стандартных средств этого языка.

Отложенная диспетчеризация

Предложенный способ реализации мультиметодов основан на так называемой *отложенной диспетчеризации* [4]. Отложенная диспетчеризация является расширением широко известного паттерна проектирования `Visitor`, или *Посетитель* [6]. Каждый

посещаемый объект должен реализовать метод `Visit()`, принимающий в качестве параметра ссылку на специальный объект – *диспетчер*. *Диспетчер* должен иметь набор методов `Dispatch()`, принимающих ссылки на конкретные типы посещаемых объектов.

В случае использования классического паттерна проектирования *Посетитель* возникает циклическая зависимость между *Посетителем* и посещаемыми объектами. Чтобы объявить интерфейс *Посетителя*, необходимо знать список посещаемых объектов. В свою очередь, чтобы объявить посещаемый объект, необходимо знать интерфейс посетителя. *Отложенный Диспетчер* позволяет частично разорвать эту связь и дает возможность объявить и реализовать посещаемые объекты до того, как будет известен интерфейс *Посетителя*.

Интерфейс *Отложенного Диспетчера* и его конкретные реализации генерируются на основе списка типов посещаемых объектов [2, 5]. *Отложенный Диспетчер* используется для определения конкретного типа посещаемого объекта. Каждый конкретный диспетчер принимает в виде шаблонного параметра шаблона [7] тип получателя информации о типе посещенного объекта, что позволяет вкладывать диспетчеры один в другой. Данный механизм позволяет определить тип n объектов, участвующих в вызове n-арного мультиметода. В качестве последнего, n+1-го уровня, указывается класс, позволяющий на основе уже известной информации о типах всех участвующих в вызове мультиметода объектах, вызвать наиболее подходящую специализацию.

Механизм вызова мультиметода схематически изображен на рис. 1.

Реализация

Реализация предлагаемого подхода достаточно сложна. Однако подавляющая часть сложного кода реализована в виде повторно используемой библиотеки классов, что значительно упрощает использование мультиметодов в прикладных приложениях. Библиотека общедоступна по адресу http://mmdd.sourceforge.net.

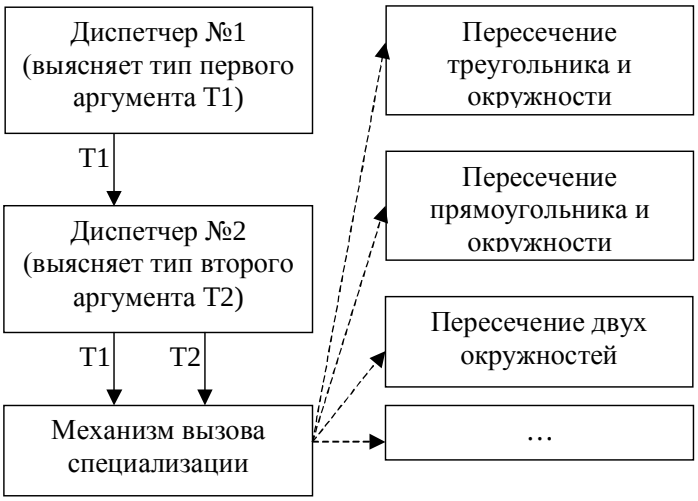

Рис. 1. Механизм вызова бинарного мультиметода

Для реализации мультиметода с использованием предлагаемой библиотеки достаточно следующего:

- все объекты в целевой иерархии должны быть *посещаемыми*, т.е. наследоваться от интерфейса `IVisitable` и перегружать виртуальный метод `Visit()`;
- библиотека должна быть сконфигурирована посредством *конфигурационного класса*, содержащего, в частности, список типов всех участвующих в вызове мультиметода объектов;

- должны быть предоставлены все необходимые специализации мультиметода;
- пользователем должна быть реализована свободная функция, вызывающая мультиметод.

Достоинства и недостатки предлагаемого подхода

Выделим достоинства предлагаемого подхода по сравнению с существующими:

- не используются никакие виды операций приведения типа;
- не используется механизм RTTI (Run-Time Type Identification);
- не используется препроцессор языка C;
- весь исходный код удовлетворяет строгим требованиям безопасности типов;
- поддерживается раздельная компиляция исходного кода;
- время вызова мультиметода константно и зависит только от арности вызываемого мультиметода;
- в процессе вызова мультиметода не используется выделение динамической памяти;
- не используются нестандартные библиотеки;
- используется только *стандартное* подмножество языка C++.

Основным недостатком предлагаемого подхода является экспоненциальный рост бинарного кода, генерируемого компилятором. Это происходит потому, что на этапе компиляции учитываются все возможные комбинации вызова мультиметода. Например, для вызова бинарного мультиметода при количестве различных типов, равном n, количество комбинаций будет равно n^2. Однако этот отрицательный фактор смягчается возможностью раздельной компиляции. Весь *экспоненциальный* код сосредоточен в пределах одной единицы компиляции и не влияет на скорость компиляции остального кода.

Заключение

Предлагаемый подход позволяет эмулировать поддержку мультиметодов в языке программирования C++. При этом, предлагаемый подход использует только *хорошие приемы* программирования на этом языке. Например, не используются такие средства как препроцессор языка C и небезопасные приведения типа в стиле C. Более того, не используется механизм RTTI, что позволяет гарантировать строгую безопасность типов.

Предлагаемый подход обеспечивает высокую скорость вызова мультиметода вследствие того, что не используется выделение динамической памяти. Кроме того, время вызова предсказуемо, константно и зависит только от арности мультиметода.

Раздельная компиляция позволяет локализовать реализацию мультиметода, не затрудняя компиляцию остального кода системы. Использование только стандартных средств языка C++ делает предлагаемый подход доступным для пользователей различных реализаций компилятора языка C++.

Литература

1. Dylan Programming: An Object-Oriented and Dynamic Language / Feinberg **N**., Keene S., Mathews R., Withington T., Mathews R. Addison-Wesley, 1996. 412 с.
2. Александреску А. Современное проектирование на C++: Обобщенное программирование и прикладные шаблоны проектирования. М.: Вильямс, 2002. 336 с.
3. Baker J., Hsieh W. Maya: multiple-dispatch syntax extension in Java // Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation, 2002. P. 270–281.
4. Shopyrin D. MultiMethods via Deferred Dispatch http://www.codeproject.com/cpp/mmvdd.asp, 2004.
5. Shopyrin D. MultiMethods in C++: Finding a complete solution http://www.codeproject.com/cpp/mmcppfcs.asp, 2004.
6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001. 368 с.
7. Vandevoorde D., Josuttis N. C++ Templates: The Complete Guide .Addison-Wesley, 2003.