УДК 681.3.06

РАБОТА С МОДУЛЯМИ В ФУНКЦИОНАЛЬНОМ ЯЗЫКЕ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ «ПИФАГОР»

Матковский И.В.
Научный руководитель – д.т.н., профессор Легалов А.И.
Сибирский федеральный университет

«Пифагор» представляет собой созданный для исследовательских целей функциональный язык программирования. В рамках данной работы рассматривается один из новейших аспектов языка – модульная система.

Разбиение на модули широко применяется при создании больших программ. Оно обеспечивает раздельную компиляцию, независимую одновременную разработку, позволяет добавлять новые модули к уже написанному программному коду и т.д. Добавление модулей не оказывает прямого воздействия на расширение существующих понятий. Это объясняется отсутствием в модуле характерных точек, предназначенных для параметризации или формирования альтернативных контекстов. Такие точки удобнее реализовывать в специальных программных объектах. Возможность расширить модуль обеспечивает единое пространство имен с уже существующими его фрагментами, что облегчает восприятие эволюционно расширяемой программы.

Промежуточное представление модуля содержит следующие элементы:

- таблица имен;
- промежуточные представления всех входящих в модуль функций;
- промежуточные представления всех входящих в модуль констант.

Таблица имен хранит ряд записей вида:
[имя объекта][тип объекта]
Имя объекта является обычной текстовой строкой. Типов объекта на данный момент три – функция, константа и вычисляемая константа.

Промежуточное представление информационного графа функций уже было описано неоднократно в других статьях о языке “Пифагор”. Промежуточное представление констант может реализовываться двояко. В случае с обычной константой, задаваемой в программном коде, выражением типа:

```
pi << 3.14159265;
```

где хранение реализуется тривиально. Такая константа будет записана в виде элемента информационного графа.

Вторая разновидность констант – константы вычисляемые. Они задаются выражениями вида;

```
x << (1,2):+;
```

Такие константы проще хранить в виде функций. От обычных функций они будут отличаться лишь тем, что их значение будет автоматически вычисляться на этапе инициализации модуля – и тогда уже заменять в памяти интерпретатора алгоритм их вычислений.

Рассмотрим порядок подключения уже существующего модуля в некоторой программе. Условно в последовательности действий можно выделить три этапа:

1. Ссылка на модуль. Содержится в начале программы, в единственном числе. Обрабатывается на этапе трансляции, проводится проверка наличия данного модуля в базе.
2. Инициализация модуля. Может производиться по факту ссылки, по факту первого обращения к модулю или по первому вызову элементов модуля. Вызывает служебную функцию-конструктор модуля; может изменять либо корректировать содержимое элементов модуля в соответствии с поступившими данными.
3. Импорт элементов модуля. Находится в теле программы и обрабатывается в процессе интерпретации. В процессе импорта запрошенные элементы выгружаются из вызванного модуля и становятся элементами новой программы.

Первый этап исполняется на этапе трансляции для проверки наличия нужных модулей. Проверка осуществляется через обращение ко всем импортированным в данную программу модулям и проверки их таблиц имен на наличие запрошенных сущностей: функций, констант и вычисляемых констант.

Инициализация модуля сводится к вызову конструктора (если таковой в модуле имеется) и сохранению копии элементов модуля в памяти интерпретатора. Функция-конструктор может изменять содержимое остальных элементов модуля в зависимости от поступивших в нее данных. Называть этот процесс изменением не вполне корректно, структурно он ближе к выбору из нескольких заготовленных заранее альтернатив. Так или иначе, содержание модуля полностью определяется уже после вызова конструктора; следовательно, эти изменения нужно каким-то образом сохранить в памяти. Возникнет необходимость в контроле сохраненных модулей. Эта задача решается легко с помощью таблицы дескрипторов. Введение этой функции позволит, к примеру, несколько раз подключить в рамках одной и той же программы один и тот же модуль, но с разными стартовыми параметрами.

После инициализации модуль сохраняется в качестве одного из элементов программы (термин “переменная” для языка «Пифагор» некорректен, но он наиболее близко отображает суть подобного сохранения). Проще всего непосредственно хранить в оперативной памяти инициализированный модуль. Возможно также помещать доработанный вариант в некую служебную поддиректорию базы данных, а в самой “переменной” сохранять лишь ссылку на него.

Последним этапом является непосредственный импорт элементов модулей. Элементами могут быть некоторые данные, функции или другие модули:

На данный момент можно выделить три способа связывания модулей

1. Внутреннее (программное). Связывание производится путем явного указания в программе перечня используемых внешних модулей. В результате формируется четкая структура взаимодействия модулей; сборка проекта целиком осуществляется без дальнейших указаний программиста. Одним из недостатков данного подхода является возможность многократного обращения к одному и тому же модулю в том случае, если на него ссылается ряд других.
2. Внешнее. В конфигурационном (проектном) файле задается перечень элементов, входящих в проект. Это помогает контролировать перечень входящих в проект элементов путем редактирования одного лишь файла и позволяет отслеживать дублирующиеся ссылки. В таком случае гарантируется, что каждый файл будет обработан интерпретатором лишь единожды. Ввиду того, что данный способ существенно облегчает введение в программу новых элементов, он лучше подходит для написания эволюционно расширяемых программ.

3. Комбинированное. Способ почти полностью аналогичен внешнему. Ряд связей все же указывается непосредственно в самой программе. Этот подход сочетает в себе достоинства первых двух и практически нивелирует их недостатки.

Наиболее простым вариантом из вышеперечисленных является первый; он и будет реализовываться в дальнейшем. В рамках стоящей задачи на данном этапе преимущества второго и третьего вариантов сильно нивелируются – для каждого конкретного подключения модуля придется создавать его отдельную копию.

Непосредственное подключение модулей может проводиться двумя путями:

1. Текстовая (препроцессорная) сборка – код модулей (элементов модулей) непосредственно вставляется в код транслируемой программы; получившаяся программа передается на трансляцию единым блоком и в дальнейшем интерпретируется как единое целое.
2. Перекрестная сборка. Интерпретатору на обработку подаются уже откомпилированные внешние модули; из их промежуточного представления извлекаются запрошенные элементы.

Очевидно, что текстовая сборка снизила бы полезность всей концепции модульной библиотеки, в таком случае, в предварительной компиляции модулей не было бы никакой практической пользы. Логичнее использовать перекрестную сборку, поскольку она не требует заново компилировать код подключаемых модулей.

Перекрестная сборка может проводиться статически, динамически или же комбинировать два подхода. Статический подход предполагает, что все используемые модули вызываются интерпретатором перед началом собственно выполнения программы. В результате получается готовая иерархическая структура, которая может сразу отправляться на выполнение. При динамическом подходе подгрузка модулей происходит непосредственно на этапе их вызова. Это позволяет обойтись без предварительной подготовки и дает возможность работать с изменяемыми вариациями модулей. Для поставленных задач динамический режим подгрузки подойдет больше.