**Создание пакетов и модулей в Perl**

В этой статье мы рассмотрим процесс создания пакетов и модулей и в качестве примера создадим один простейший модуль и пакет.

**Intro**

Защищенность и модульность - два великих принципа программирования. Perl обеспечивает их выполнение, предоставляя возможность разбивать программу на полуавтономные фрагменты так, что программисту не надо беспокоиться о конфликтах между ними и остальной частью программы. Для деления программы на независимые фрагменты используются пакеты Perl, которые создают непересекающиеся области имен (namespaces). Что такое область имен? Это часть программы со своей собственное областью видимости глобальных идентификаторов - другими словами, она функционирует как частная территория программиста.

На самом деле в Perl нет такой вещи, как "область видимости глобальных идентификаторов", - любая такая область ограничивается неким пакетом. Создавая пакет, вы получаете некую гарантию того, что ваш код не смешается с переменными и подпрограммами другого фрагмента. Это позволяет организовывать код, предназначенный для многократного использования, в виде пакетов.

Кроме пакетов существуют также модули Perl. Моудли - это пакеты, организованные специальным образом. Их можно загружать и интегрировать с конкретной программой. В этой статье пойдет речь о создании модулей и пакетов.

**Пакеты**

Пакет можно представить в виде юнита Delphi. Код, помещаемый в пакет, может размещаться во внешнем файле, в нескольких файлах, хотя несколько пакетов могут размещаться в одном файле (что невозможно сделать в дельфийском юните). Переключаться между различными пакетами внутри файла нужно с помощью команды package. Давайте создадим простой пакет и сохраним его как package1.pl:

package package1;

BEGIN { }

sub subroutine1 {print "Hello!\n";}

return 1;

END { }

Команда package начинает новый пакет package1. Обратите внимание на подпрограммы BEGIN и END. Первая подпрограмма выполняется сразу же после загрузки пакета. Поэтому в неё обычно помещают инициализирующий код. Хотя вернее было бы утверждать, что подпрограмма BEGIN выполняется как только интерпретатор доходит до неё, т.е. до окончания загрузки пакета. А подпрограмма END выполняется при завершении работы интерпретатора и может содержать код, выполняющий заключительные оперпации (например закрытие открытых файлов. Подпрограммы BEGIN и END вызываются неявным образом (более того, вам никогда не удастся явно вызвать BEGIN: интерпретатор уничтожает её сразу же после использования). Именно поэтому эти подпрограммы состоят из заглавных букв, и ключевое слово sub для них можно не указывать.

Обратите внимание на подпрограмму subroutine1. Её можно вызывать в пределах кода, использующего пакет. Кроме того, стоит обратить внимание на команду return, расположенную вне каких либо подпрограмм, - она возвращает значение "истина" после загрузки пакета, показывая таким образом, что пакет готов к работе (на самом деле возвращается последнее значение, вычисленное в теле пакета, поэтому часто вместо строки return 1 ставится просто единица).

Что бы использовать в программе код пакета, необходимо поместить в сценарий команду require:

require "package1.pl";

Теперь можно ссылаться на идентификаторы пакета package1, отделив его имя от идентификатора двумя двоеточиями "::". Раньше в роли разделителя был апостроф (будьте внимательны, потому что этот разделитель используется и сейчас). Но теперь Perl следует стилю C++ и использует "::". Вот пример вызова подпрограммы subroutine1 из пакета package1:

require "package1.pl";

package1::subroutine1();

Как результат работы этой программы будет выведена надпись "Hello!". Можно также в пакеты помещать другие идентификаторы, например переменные:

package package1;

BEGIN { }

$var1=1;

sub subroutine1 {print "Hello!\n";}

return 1;

END { }

Использовать эту переменную легко. Достаточно подставить символ "$" перед конструкцией вызова. Пример:

require "package1.pl";

$package1::var1;

Обратите внимание, что символ "$" ставится перед именем пакета, но НЕ ставится после :: перед var1. Однако таким способом невозможно добраться до переменных, описанных с ключевым словом my: они обладают лексической областью видимости и доступны только внутри модуля.

При обращении к идентификаторам можно опускать имя пакета, и тогда будет использован пакет main (строка $::var1 эквивалентна $main:var1).

Если в программе нужно довольно часто обращаться к идентификаторам из пакетов, то код становится большим и малопонятным. Что бы решить эту проблему нужно использовать модули. При использовании модулей можно экспортировать имена, указанные в модуле в текущюю область имен.

**Модули**

Модули - это пакеты, оформленные в отдельных файлах, у которых имена последних совпадают с именами модулей и имеют расширение pm. По соглашению Perl определяет, что имя модуля начинается с заглавной буквы. Код, содержащийся в модуле, в отличие от "пакетного" кода, может экспортировать глобальные имена в текущюю область глобальных имен. Это означает, что при обращении к идентификатору не нужно указывать имя пакета.

Рассмотрим пример. Создайте модуль с именем Module1 и cохраните его в файле Module1.pm. В коде подпрограммы BEGIN, выполняемом при загрузке модуля, будем использовать стандартный модуль Exporter, что бы экспортировать имя подпрограммы subroutine1:

package Module1;

BEGIN {

use Exporter ();

@ISA = "Exporter";

@EXPORT = "&subroutine1";

}

sub subroutine1 {print "Hello!\n";}

return1;

END { }

Для использования модуля в программе нужно подключить его с помощью команды "use" (он будет включен в момент компиляции). Если же подключить модуль командой require то модуль подключится в момент исполнения сценария. Пример:

use Module1;

subroutine1();

В результате выполнения этого кода будет выведена строка "Hello!".

**Outro**

Эта статья не претендует на полноту описания модулей и пакетов. Существует ещё много, чего вы не узнали из этой статьи: пакеты можно вкладывать друг в друга, разрешать экспортировать определенные имена и не экспортировать их по умолчанию и даже вызываь несуществующие подпрограммы. Но это тема огромной главы книги, если не всей книги.

При подготовке статьи были использованы материалы из книги "Perl. Специальный справочник" (автор Стивен Холзнер, издательство "Питер", 2001 г.)