Чего ждать от Perl 6

Андрей Шитов

Perl готовится сделать новый рывок и оставить конкурентов позади. До круглой цифры счетчику версий осталось совсем немного, всего 0.1.2 версии.

Про Perl 6 впервые начали говорить в 2000 году, когда было принято решение переписать существующий язык с чистого листа. Исторически сложилось так, что формального стандарта никогда не существовало. Стандартом пятой версии фактически был язык, воплощенный в интерпретаторе. Желание иметь четкие правила – одна из предпосылок к созданию нового языка.

Кроме того, требовались изменения, чтобы, с одной стороны, очистить язык от устаревших конструкций (например, GLOB), а с другой – добавить «современности» (например, включив в язык полноценную поддержку классов).

Часть 1. До языка

К сожалению для разработчиков и к счастью для стандарта, после шести лет еще не существует ни окончательных правил, ни полноценного компилятора. Тем не менее, уже сегодня язык можно применять на практике (хотя так будут поступать разве что истинные энтузиасты), а уж познакомиться с языком просто необходимо.

Общее направление задал создатель Perl Ларри Уолл: в статьях под общим названием «Откровения» (Apocalypses) описано, каким он видит будущий язык.

Позже появился еще один цикл «Толкования» (Exegeses), в которых Дамиан Конвей поясняет «Откровения» на примерах. И, наконец, существует начатый Эллисон Рэндал набор

«Конспектов» (Synopses); они по сути являются спецификацией, от которой отталкиваются разработчики компиляторов.

Одна из разработчиков (да, это женщина!) нового компилятора Одри Танг (Audrey Tang) изобразила на условном графике эволюцию Perl 6. Развитие временами напоминает хаос, тем не менее, надежда увидеть работающий Perl 6 есть.

Perl 6 и Perl 5

Синтаксис пятой и шестой версий не совместим друг с другом. Поэтому, чтобы начать изучать Perl 6, не обязательно предварительно осваивать Perl 5. Однако программистам, знакомым с пятой версией, будет проще понять многие конструкции, которые в новую версию либо полностью перешли из предыдущей, либо изменились только косметически.

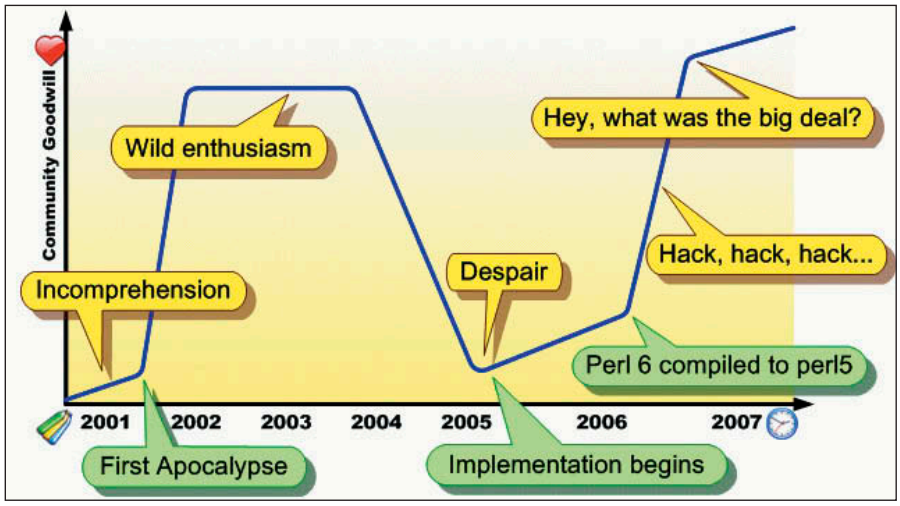
Perl 5 и Perl 6 не просто разные версии одного языка. Правильнее будет сказать, что это два разных языка, объединенных одной идеей. Нынешние компиляторы не способны выполнить программу другой версии. Тем не менее изначально предполагалось, что рабочая версия нового компилятора станет универсальной.

Кстати, предыдущий «апгрейд» языка с четвертой версии на пятую тоже требовал от программистов дополнительных усилий из-за некоторых несовместимостей.

Архитектура

«Прошлый» Perl всегда был интерпретируемым языком. Это часто давало повод противникам говорить о невозможности применять язык в критических по времени приложениях. Наличие mod_perl и компиляторов perl2exe не спасает положения, поскольку, например, для программирования под mod_perl требуется учитывать специфические ограничения, а perl2exe редко способен сгенерировать работающий код для программы, более сложной, чем «Hello, World!».

В Perl 6 с самого начало было решено изменить традиционный подход и применить схему «программа – компилятор – байт-код – виртуальная машина». Эта схема не новая, и к 2000 году уже работала с языком Java, а чуть позже в технологии .NET.



Условная кривая развития Perl 6. (Рисунок Одри Танг)

Новый цикл разработки программ на Perl 6 дает сразу два преимущества. Во-первых, во время выполнения теперь не требуется интерпретировать исходный код, вместо этого виртуальная машина выполняет предварительно скомпилированный байт-код (идеология Java в чистом виде). Во-вторых, появилась возможность создавать программы из частей на разных языках, для которых созданы трансляторы в байт-код (идеология .NET). На первый взгляд может показаться, что не было смысла создавать собственный формат байт-кода, а воспользоваться, например, правилами .NET. Но основные языки, с которыми работает .NET, – языки с сильной типизацией. Байт-код же, используемый для Perl 6, сконструирован с расчетом на динамические языки без фиксированных типов данных, подобные Perl, например, Python и Ruby.

Наличие промежуточного байт-кода не отменяет преимущества новой версии как традиционного скриптового языка. Программу всегда можно модифицировать на ходу, выполняя ее с командной строки и не задумываясь о том, что происходит внутри компилятора.

Parrot, PASM, IMC, PBC, Ponie, PIR, PUGS, PIL, Perl6::*

История развития за последние шесть лет, даже несмотря на наличие генплана, испещрена названиями и аббревиатурами; часть из них уже потеряли свою актуальность.

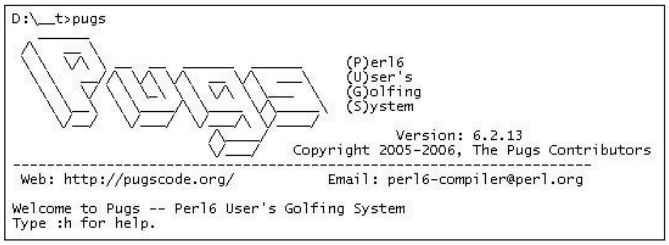
Parrot – это название упомянутой выше виртуальной машины. На ранней стадии использовался собственный ассемблер (Parrot assembler, PASM), который содержит набор инструкций, очень похожих на те, которые применяются в обычном ассемблере для физических RISC-процессоров. Все операции выполняются с операндами в виртуальных регистрах разных типов: например, в одних регистрах хранятся целые числа, в других строки. Один из типов необычен (как и его название Parrot Magic Cookie, PMC) и предназначен специально для поддержки объектно-ориентированного программирования. Байт-код сохранялся в файле с расширением .pbc (pre-compiled bytecode).

Поскольку число регистров ограниченно, программировать непосредственно на ассемблере PASM неудобно (конечно, прежде всего разработчикам компиляторов), и появились сопутствующий язык IMC (Intermediate Code) и компилятор IMCC (Intermediate Code Compier). IMC представляет собой надстройку над ассемблером и разрешает пользоваться произвольными переменными, поэтому нехватка регистров перестает ощущаться. Сегодня вместо IMC используется PIR (Parrot Intermediate Representation).

Ponie (еще одна тупиковая ветвь в развитии Perl 6) – это попытка создать компилятор в байт-код для программ, написанных на Perl 5.

Библиотека CPAN содержит набор модулей Perl6::*, которые были написаны, чтобы поэкспериментировать с синтаксисом шестой версии, имея компилятор пятой. Пользоваться этими модулями сегодня особой необходимости нет.

В состав Parrot входит около десятка компиляторов, создающих байт-



Приветствие, выводимое при запуске Pugs

код из программ на разных языках (иногда экзотических типа Ook!), в том числе экспериментальный компилятор Perl 6.

Компилятор в составе Parrot более не развивается, ему на смену пришел проект Pugs (Perl 6 User's Golfing System), которым занимается Одри Танг. Pugs тоже внес свой вклад в список сокращений, например, создав собственный формат PIL (Pugs Intermediate Language).

Установка Perl 6

Разбираться в месиве форматов необязательно, достаточно установить последнюю версию Pugs. Правда, радужная картина немного меркнет, когда узнаешь, что Pugs написан на языке Haskell, и для сборки необходим работающий компилятор. (А тем, кто захочет попробовать в действии новые регулярные выражения, возможно, потребуется и установленный Parrot.) Компилятор доступен на сайте www.haskell.org/ghc как в исходных кодах, так и в бинарных дистрибутивах для некоторых платформ. Самостоятельная сборка компилятора GHC происходит крайне долго, хотя и гладко.

Итак, если для вашей платформы не нашлось бинарной версии Pugs, необходимо последовательно выполнить две процесса: установить компилятор GHC, после чего собрать Pugs.

Ссылки на исходные и бинарные коды находятся на сайте проекта www.pugscode.org.

Процесс сборки из исходных кодов стандартный для модулей Perl:

```
perl Makefile.PL
make
make install
```

Примечание: исходный код Pugs размещен на сайте CPAN, но он не имеет отношения к модулям Perl6::*, о которых говорилось в предыдущем разделе.

Во время подготовки статьи для тестирования программ я пользовался двумя версиями Pugs под Windows XP SP2 и Linux Red Hat 9. На каждой системе установлены GHC 6.6, Parrot 0.4.6, Pugs 6.2.13 и Perl 5.8.8.

Кстати, при установке Pugs просит самую последнюю версию компилятора GHC, сообщая, что с более ранней скомпилировать-то пока удастся, но скорость работы будет низкой, а в будущем старая версия GHC вообще не подойдет.

Компилятор Haskell под Windows установлен из msi-файла (необходим один дистрибутив размером 37 Мб, находящийся на официальном сайте).

На второй системе получить желаемый набор удалось не сразу. В первый раз компилятор GHC собрался из исходных кодов без ошибок (за пару часов), но затем при установке Pugs конфигуратор сообщил, что не хватает модуля mtl в составе GHC:

```
*** Could not load the "mtl" package in your GHC installation.
```

Впрочем, далее в сообщении об ошибке следует разъяснение о том, что дополнительно необходимо поставить комплект библиотек extralibs.

Итого, при сборке компилятора Haskell из исходных кодов нужны два файла, в данном случае ghc-6.6-src.tar.bz2 и ghc-6.6-src-extralibs.tar.bz2. Они находятся на том же сайте в разделе Source Distribution. Распаковав их в общий каталог и заново собрав GHC (еще два часа), удалось получить все необходимое для сборки Pugs.

Часть 2. Язык

Самостоятельное изучение Perl 6 – сегодня занятие, больше похожее на исследовательскую работу. Теоретические источники знаний подборка Synopses, практические каталоги с примерами, входящими в поставку Parrot (languages/perl6/examples) и Pugs (examples/).

В оставшейся части статьи бегло рассмотрены основные моменты Perl 6. Предполагаю, что вы знакомы с пятой версией, в этом случае чувство новизны языка ощущается сильнее.

Все приведенные примеры кода проверены в Pugs 6.2.13 на двух системах: под Windows и Linux. Запуск программ на исполнение аналогичен тому, как это происходит в Perl 5. Если код записан в файле, передайте его аргументом при вызове pugs:

```
> pugs programme.p6
```

Короткие однострочные программы допустимо записывать непосредственно в командной строке:

```
> pugs -e'print "This is Perl 6"'
```

Кстати, для запуска pugs удобно сделать псевдоним p6 или perl6.

Наконец, на платформе UNIX в файле с программой можно указать путь к компилятору:

```
# !/usr/local/bin/pugs
```

и сделать файл исполняемым:

```
> chmod 0755 programme.p6
> ./programme.p6
```

К сожалению, объем журнальной статьи не позволяет описать все детали новых возможностей языка. Новшеств много, и они находятся на самых разных уровнях от способа записи скобок при вызове функции до идеологии объектно-ориентированного программирования. Хотя я отобрал для обзора наиболее интересные изменения, кому-то все равно может показаться, что подборка несколько субъективна.

Все приведенные примеры содержатся в архиве, доступном по адресу: http://perl6.ru/publish/samag/code.zip.

Вывод на печать

Прежде чем рассматривать особенности Perl 6, необходимо познакомиться с механизмом вывода на печать, поскольку он потребуется в каждом примере.

Стандартная функция print() по-прежнему доступна, но в Perl 6 появился метод say(), действие которого аналогично работе функции print(), но всегда завершается переводом строки. Новый способ вывода особенно удобен в отладочных сценариях, когда вписывать переводы строки вручную утомительно.

```
# say.p6
say "This is Perl 6";
say 123;
say(8 * 9);
```

Я не случайно назвал say() методом, поскольку им можно пользоваться не только как функцией, но и как методом объекта, например:

```
# say-method.p6
"This is Perl 6".say;
123.say;
(8 * 9).say();
```

Вторая программа напечатает то же самое. Метод say() доступен для любых встроенных типов данных, в том числе для чисел. Обратите внимание, что скобки после имени метода необязательны. Однако перед открывающей скобкой не должно быть пробела:

```
'error'.say ();
```

Такая запись приведет к ошибке:

```
$ p6 say-method.p6
```

```
***
    Unexpected "();"
    expecting comment, operator, statement modifier,
    ";" or end of input at say-method.p6
    line 7, column 13
```

Комментарии оформляются так же, как и в Perl 5, символом #.

Поддержка юникода

Исходный код программ на Perl 6 по умолчанию считается написанным в юникоде. Поэтому можно смело пользоваться любыми символами из нелатинских кодировок:

```
# unicode.p6
say "Превед!";
say "β-sitostérol";
say "鍵 盤 迷 你 大 革 命!"
```

Если выполнить эту программу с командной строки в стандартной оболочке, символы отобразятся неправильно; чтобы убедиться в работоспособности, сохраните вывод в файл.

Переменные

В Perl 6 по-прежнему применяются сигилы (sigils) для обозначения типа структуры переменной: $ – для скаляров, @ – для списков и % – для хешей. Однако теперь при обращении к отдельным элементам массивов и хешей (а отдельные элементы чаще всего бывают скалярами) сигил изменять не требуется. Например, обращение к элементу списка выглядит как @array[5], в то время как раньше приходилось писать $array[5]. Иными словами, переменная всегда сохраняет свой префикс-сигил.

```
# variables.p6
my $string = 'abcde';
my @array = ('alpha', 'beta', 'gamma');
my %hash = {'alpha' => 'A', 'beta' => 'B', 'gamma' => 'C'};
say $string;            # abcde
say @array[1];          # beta
say %hash{'gamma'};     # C
```

Связывание переменных

Связывание (binding), или создание ссылок, осуществляют операторы «:=» и «::=». Они создают синоним переменной, указанной справа от оператора. Любое обращение к синониму эквивалентно обращению к оригинальной переменной:

```
# binding.p6
my $a = 123;
my $b := $a;
$b = 456;
say $a; # 456
```

Хотя в последней строке происходит обращение к переменной $a, на экране появится значение 456, которое присвоено через ссылку $b.

Различие между формами с одним и двумя двоеточиями заключается в моменте, когда выполняется связывание. В первом случае оно происходит на этапе выполнения программы, во втором – на этапе компиляции.

Это также сказывается и на размере промежуточного кода.

Например, если приведенную программу оттранслировать в промежуточный PIR-файл (указав опцию -CPIR), получится код длиной 91 049 байт, а та же программа с оператором «::=» окажется чуть короче (90 897 байт). Не стоит пугаться таких размеров промежуточного кода, потому что в нем собрано много определений операторов, которые данной программой не используются.

Конкатенация строк

«Складывание» строк теперь выполняет не оператор «точка», а тильда:

```
# concatenation.p6
my $before = 'This is';
my $after = 'Perl 6.';
say $before ~ ' ' ~ $after;
```

Одинарные и двойные кавычки в строках имеют тот же смысл, что и в Perl 5.

Операторы «//» и «//=»

Бинарный оператор «//» (не путайте с началом однострочного комментария) принимает два операнда и возвращает первый из них, если он определен, и второй, если не определен первый.

```
# double-slash.p6
my $a = "a";
my $b = "b";
my $c;
say $b // $a; # b
say $c // $a; # a
```

При первом вызове метод say() получает значение переменной $b (поскольку она определена), а второй $a (потому что значение у переменной $c отсутствует).

Оператор «//=» выполняет то же действие, но над единственной переменной: вызов «$a //= $b» равнозначен «$a = $a // $b».

Интервальные условия

В Perl 6 операторы сравнения допускается объединять в цепочку, что позволяет создавать компактные записи, например, для проверки попадания значения переменной в заданный интервал:

```
# interval.p6
my $x = 5;
say "yes" if 0 < $x < 10;
if 0 < $x < 10 {say "yes"}
```

Кстати, из второго блока сравнений видно, что теперь необязательно использовать скобки в условии.

Контексты

Perl 6 позволяет вручную управлять контекстом, в котором происходит работа с переменной; в некотором смысле это подобно преобразованию типов.

Контекст устанавливается с помощью унарного оператора:

```
# contexts.p6
my @array = (5..10);

# Строковый контекст
say ~@array; # 5 6 7 8 9 10

# Числовой контекст (возвращается размер массива)
say int @array; # 6
say +@array;   # 6

# Два модификатора контекста
say ~ hash @array; # напечатает содержимое в два столбца

my $value = 100;

# Булевый контекст
say ?$value; # 1 (то есть истина)
```

Функции

Ключевое слово для объявления функции sub не изменилось. А механизм передачи параметров стал нагляднее. Например, при объявлении функции возможно указать именованные аргументы.

При вызове нужно либо соблюдать порядок объявления, либо явно указывать имена.

```
# sub-args.p6

callme(10, 20);
callme(second => 7, first => 8);
sub callme ($first, $second)
{
      say "\$first = $first, \$second = $second";
}
```

Еще раз отмечу, что при вызове функции скобки нужно ставить вплотную к имени, иначе произойдет ошибка:

```
Extra space found after &callme (...) –
did you mean &callme(...) instead?
```

В объявлении наличие или отсутствие пробела не имеет значения.

Одновременно с именами задаются и структурные типы формальных параметров (это было возможно и ранее, но не так наглядно).

Параметры функции могут содержать модификаторы (traits), изменяющие их поведение. В частности, модификатор is rw дает возможность перезаписывать переданный аргумент:

```
# sub-rw.p6
my $string = "before";
callme($string);
say $string; # after
sub callme ($value is rw)
{
      $value = "after";
}
```

Попытка изменить аргумент без такого модификатора будет пресечена компилятором:

```
*** Can't modify constant item: VRef <Scalar:0x1c6cfc4>
    at sub-rw.p6 line 9, column 9-25
```

Чтобы произвольно изменять значение, требуется указать модификатор is copy.

При вызове функции по умолчанию не происходит «сворачивания» всех переменных в общий массив. Например, теперь удается безболезненно передать в функцию два массива, нигде явно не указывая их длину:

```
# sub-slurp.p6
my @odd = (1, 3, 5);
my @even = (2, 4, 6);
view(@odd, @even);
sub view (@a, @b)
{
      say @a;
      say "&";
      say @b;
}
```

Необязательные параметры (которые должны идти последними в списке) помечают вопросительным знаком. Дополнительно можно указать значение по умолчанию:

```
# sub-optional.p6
callme(1, 2); # 1, 2
callme(3);    # 3, 4
sub callme ($a, $b? = 4)
{
      say "$a, $b";
}
```

И наконец Perl 6 позволяет объявлять анонимные функции с помощью стрелки:

```
# sub-anonymous.p6

my $anonymous = -> ($value)
{
      say $value * 2;
}
$anonymous(10); # 20
```

В этом примере переменной $anonymous присвоен указатель на безымянную функцию, после чего происходит вызов посредством этого указателя.

Перегрузка функций

Ключевое слово multi разрешает перегружать функции, то есть создавать несколько функций с одинаковым названием, но разным типом или числом аргументов.

```
# sub-reload.p6
multi sub action ($scalar)
{
      say "scalar";
}
multi sub action ($scalar, $scalar)
{
      say "two scalars";
}
multi sub action (@array)
{
      say "array";
}
action(10); # scalar
action(10, 11); # two scalars
my @arr = (1, 2);
action(@arr); # array
```

Перегрузка операторов

Возможность перегружать не только функции, но и операторы явно придется по вкусу любителям C++. Perl 6 позволяет с помощью ключевых слов prefix, infix и postfix определять собственную семантику предопределенных операторов либо создавать новые:

```
# xfix.p6

multi infix:<+> ($a, $b)
{
      return $a - $b;
}
say 10 + 20; # -10
sub postfix:<@> ($power)
{
      2 ** $power;
}
say 8@; # 256
```

gather и take

В Perl 6 появилась удобная пара gather/take, которая помогает избежать вспомогательного массива, когда нужно сохранять промежуточные результаты в циклических операциях.

```
# gather.p6
say gather
{
   for 1..5 -> $c
   {
      take $c;
   }
}
```

Каждый вызов take добавляет новое значение в массив, возвращаемый блоком gather, и программа напечатает строку 12345. (Кстати, стрелка –> здесь является не чем иным, как началом анонимной функции.)

given, when и default

Ключевые слова given, when и default предназначены для организации конструкции типа switch в C:

```
# given.p6
my $x = 'y';
given ($x)
{
      when "a" {say 'First letter'}
      when "b" {say 'Second letter'}
      default {say "Is '$x'"}
}
```

Блоки when могут содержать и более сложные конструкции, например, сопоставление с регулярным выражением:

```
given ($x)
{
      when /<[a-z]>/ {say 'isalpha'}
}
```

Программа напечатает isaplha, поскольку буква «y» совпадает с символьным классом <[a-z]> в регулярном выражении.

Новые регулярные выражения

В документации на Perl 6 большой раздел посвящен новому синтаксису (не будет ошибкой сказать – новой идеологии) регулярных выражений. Кроме правил (regexps) теперь существуют грамматики (grammars), их объединяющие. Например, ранняя версия компилятора Perl 6, входящая в состав Parrot, при первом запуске создавала файл Perl6grammar.pm, описывающий синтаксис Perl 6, записанный в виде грамматики Perl 6. Подробное рассмотрение новых регулярных выражений требует отдельной статьи.

Классы и роли

Классы и роли – одно из самых существенных нововведений в Perl 6. Формально поддержка ООП была и в Perl 5, но «классы» являлись просто хешами с некоторыми дополнительными возможностями. Механизмы ООП в Perl 6 проще для понимания и образуют намного более стройную модель. Чтобы научиться применять классы, программистам, знакомым с другими объектными языками, достаточно познакомиться с синтаксисом, присущим Perl 6. Для определения класса служит ключевое слово «class»:

```
# class.p6
class Alphabet
{
}
```

Объекты класса создаются с помощью оператора new:

```
my $abc = new Alphabet;
```

Члены-данные объявляют, используя has. Точка перед именем является признаком того, что переменная объявляется как открытая (public в общепринятой терминологии); отсутствие точки делает ее закрытой (private):

```
class Alphabet
{
      has $.Name;
      has $Length;
}
my $abc = new Alphabet;
$abc.Name = 'Latin';     # синтаксически верно
#$abc.Length = 26;       # ошибка
```

При создании объекта возможно инициализировать члены-данные:

```
my $abc = Alphabet.new(Name => 'Latin', length => 26);
say $abc.Name;
```

Классы могут содержать методы; они так и объявляются с ключевым словом «method»:

```
class Alphabet
{
      has $Name;
      has $Length;
      method Info
      {
                return "Aplhabet '$.Name' contains ↵
                          $.Length letters.";
      }
}
```

Обратите внимание: хотя теперь обе переменные объявлены закрытыми, при обращении к ним из методов класса все равно необходима точка: $.Name. Конструктор и деструктор – это методы с именами BUILD и DESTROY.

Классы можно наследовать, причем в Perl 6 допустимо множественное наследование. Чтобы создать производный класс, нужно указать базовый после ключевого слова «is»:

```
class Characters is Alphabet
{
}
my $chars = new Characters;
say $chars.Info();
```

Разрешено и множественное наследование:

```
class Unique;
class Characters is Alphabet is Unique
{
}
```

В Perl 6 появилось новое понятие ролей. В других языках поведение, аналогичное действию ролей, реализуют либо абстрактные классы, либо интерфейсы. По сути, роли – это такие классы, которые запрещено инстанциировать (то есть создавать отдельные экземпляры этого класса). Чтобы «приписать» классу некоторую роль, нужно создать производный класс, воспользовавшись ключевым словом «does» вместо «is».

```
role HaveName
{
      has $.Name;
      method GetName
      {
                return $.Name;
      }
}
class NamedAplhabet does HaveName
{
}
my $abc = NamedAplhabet.new(Name => 'English');
say $abc.GetName();
```

Роли могут быть как полноценными классами, содержащими и данные, и методы, так и абстрактными интерфейсами. Для объявления абстрактных методов служит ключевое слово «...» (три точки), которое, к сожалению, не реализовано в текущей версии Pugs.

Perl 6 на бумаге

Книг, посвященных Perl 6, пока исключительно мало.

Первая Perl 6 Essentials выпущена издательством O'Reilly летом 2003 года (ISBN 0-596-00499-0) и переиздана через год под измененным названием Perl 6 and Parrot Essentials (ISBN 0-596-00737-X). Второе издание переведено на русский язык: «Perl 6 и Parrot: справочник». М.: «Кудиц-образ», 2005. ISBN 5-9579-0086-9. Обе книги написаны Э. Рэндалом, Д. Сугальски и Л. Тёчем.

Во время издания этих книг разговор о Perl 6 не мог обойтись без более или менее детального описания Parrot. Поэтому значительная часть книги посвящена языку ассемблера Parrot.

К сожалению, по этим книгам нельзя научиться программировать на Perl 6, поскольку описанные языковые конструкции либо остались нереализованными, либо радикально изменились. Эти книги теперь можно назвать «историческими» или «идеологическими».

В 2005 году вышла в свет книга Скота Вальтерса Perl 6 Now. The Core Ideas Illustrated with Perl 5. Примеры кода часто используют модули Perl6::* и, возможно, не будут работать в «настоящем Perl 6» в том виде, в каком они опубликованы (впрочем, от этого не застрахованы и примеры из моей статьи). Примеры кода из книги доступны на сайте www.perl6now.com.

Perl 6 в Интернете

На русском языке:

1. http://perl6.ru – блог, в котором я описываю процесс освоения Perl 6. Первая запись на этом сайте появилась в 2003 году.
2. http://real.perl6.ru – первый в России сайт, работающий под управлением Perl 6. Здесь есть несколько крошечных примеров того, как можно написать CGI-приложение на Perl 6.
3. http://www.dklab.ru/chicken/perl6 – перевод фрагментов первых версий «Откровений» Ларри Уолла.
4. http://www.parrotcode.ks.ua/docs – перевод документации ранней версии Parrot.

На английском языке:

1. http://dev.perl.org/perl6 – сайт Perl Development: Perl 6, который является отправной точкой. Здесь же размещены основные документы, описывающие будущую структуру языка: Apocalypses, Exegeses и Synopses.
2. http://perlcabal.org/syn – на этой странице, названной Official Perl 6 Documentation, содержится хорошо структурированный сборник основных «правовых» документов.
3. http://www.parrotcode.org – Parrot Virtual Machine. Исходные коды и другая информация про Parrot.
4. http://www.pugscode.org – сайт Pugs.
5. http://perl6.org, http://perl6.info – архивы рассылок perl6-users, perl6-languages и других, посвященных языку.
6. http://www.programmersheaven.com/2/Perl6-FAQ – свежий FAQ с примерами кода.
7. http://www.cpan6.org – находящийся на зачаточной стадии новый CPAN, ориентированный на новый язык.
8. http://en.wikipedia.org/wiki/Perl_6 – Perl 6 в «Википедии».