

Программирование на Perl

Преподаватели

Владимир Перепелица

Известный разработчик асинхронно-событийных модулей и приложений на Perl.

Технический директор Облака@Mail.ru.



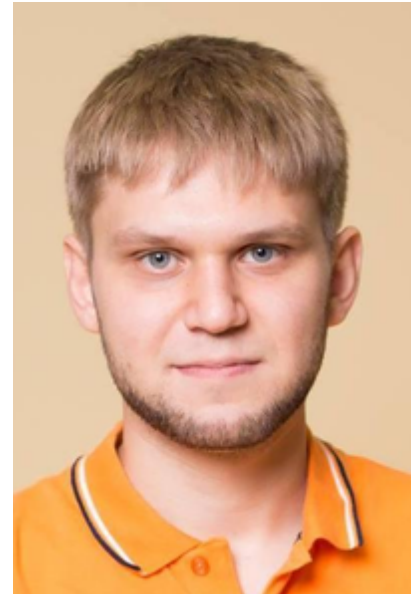
Преподаватели

Вадим Пуштаев

Разработчик проекта `поиск@mail.ru`.

Любит Perl как современный язык программирования.

Активно отвечает на вопросы на `stackoverflow`



Преподаватели

Николай Шуляковский

Экс ведущий разработчик проекта Почта@mail.ru

Заместитель технического директора проекта
МойМир@mail.ru





Отметьтесь на портале!

Содержание

1. **Цель курса**
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Что будем учить?

Язык Perl и его область применения

Основы WEB программирования и роль Perl-a в нём

Написание асинхронных программ на Perl

Метаязык XS

Основы работы с базами данных из языка Perl

Как будем учиться?

11 лекционных занятий

- на каждом занятии будет домашнее задание
- проверка домашних заданий проходит в онлайн, и защищается у того преподавателя, который вёл лекцию
- время на выполнение домашних заданий 1 неделя

1 хакатон

- командная работа над разными компонентами одного приложения

Контрольный рубеж №3 (Экзамен)

- защита домашних заданий по PerlXS и асинхронному программированию
- ответы на вопросы по всем домашним заданиям которые не было защищено или защищены не на максимальный балл

Подготовка к лекциям

У нас есть методичка!

Перед каждой лекцией студент обязан просмотреть материалы по лекции из методички, подготовить вопросы по нему.

Так же принимается фидбек по методичке:

- можно присылать материалы, которые необходимо в неё добавить
- исправление ошибок

Как оцениваются результаты?

Максимальное количество баллов, которое можно получить за этот курс - 100

За все домашние задания - 80 (10,10,7,8,7,7,7,8,8,8)

Хакатон - 20

Для получения диплома необходимо набрать 70 баллов.

Все студенты получающие сертификат будут приглашены на собеседование, для прохождения стажировки.

Содержание

1. Цель курса
2. **История создания языка Perl**
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

История создания языка Perl

18 декабря 1987г. — вышла первая версия языка программирования Perl

Это была универсальная утилита для работы с текстом.



История создания языка Perl

Развитие на то время было мотивированно тем, что программисты из разных стран отправляли Ларри Уоллу предложения по модернизации языка и доработкам

1988 году вышла версия 2.0

1989 году вышла версия 3.0

1991 году вышла версия 4.0

1994 году появляется знаменитая 5 версия языка Perl

При её подготовке было многое переосмыслено, и почти полностью переписано. Он стал модульным в нем появились зачатки ООП

"Настоящее величие в том, сколько свободы вы даёте другим, а не в том, как вам удаётся заставлять других делать то, что вы хотите" - Ларри Уолл

История создания языка Perl

До выхода 5 версии Ларри делал Perl для сообщества и по просьбе сообщества

А начиная с версии 5.0 он стал придерживаться позиции, что теперь этот язык должно делать сообщество само для себя и под свои нужды

История создания языка Perl

Происходит изменение акцентов развития языка

Все занялись написанием модулей

Perl развивался неспешно

Новые версии появлялись только, как тестовые

Параллельно началась разработка perl 5.10

История создания языка Perl

До 2005 года perl занимал в нише web-программирования лидирующее положение

Руководствуясь тем, что PHP было мало, а Java была уже перебором выбор останавливался на Perl

История создания языка Perl

С 2005 года перл начал терять свои позиции в области веб разработки, особенно под давлением PHP

Проигрыш языку, который создавался именно для web-разработки, был логичным

Не было сравнимого по своим функциям IDE

Стоимость разработчика на Perl была намного выше, чем на PHP

Порог входа в PHP ниже

Некрасивые ошибки вида "Internal Server Error"

История создания языка Perl

perl6 в истории развития языка в целом:

Сайт perl6.ru появился в 2003 году

Что то вменяемое можно было написать только к 2010 году

В 2014 году стало все гораздо лучше, но в продакшене мы его еще не увидели

История создания языка Perl

perl6 в истории развития языка в целом:

Сайт perl6.ru появился в 2003 году

Что то вменяемое можно было написать только к 2010 году

В 2014 году стало все гораздо лучше, но в продакшене мы его еще не увидели

25 декабря 2015 года состоялся релиз компилятора Rakudo 2015.12

История создания языка Perl

Как это не парадоксально, но perl6 сыграл очень большую роль в развитии perl5

Внедрение «современности», а именно полноценная поддержка классов была успешно реализована в perl5

История создания языка Perl

Итого на сегодняшний момент мы имеем хорошо зарекомендовавший себя язык

Огромную быстро растущую библиотеку

Большое, активное сообщество

Язык Perl6 и его отличный прототип, который вносит свои коррективы в развитие perl5

А так же идею сделать perl7, который быстро попадёт в продакшин, основываясь на опыте создания perl6 и надобности perl5

Высокие зарплаты

Содержание

1. Цель курса
2. История создания языка Perl
3. **Сравнение производительности**
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Сравнение производительности

Самое слабое место - это математика.

По отношению к языку C проигрыш до 20-30 раз, иногда и больше.

По отношению к js около 2-10 раз

По отношению к python примерно на 10%

По отношению к PHP примерно на 4%

Тяжёлые математические расчеты можно уносить в C + XS или lua, XS+FFI, PDL

Сравнение производительности

Регулярные выражения (парсеры)

Perl выигрывает в этом отношении у всех динамических языков и по скорости разработки и по производительности

По сравнению с языком C, perl выигрывает по скорости разработки и немного уступает по производительности

Написать регулярное выражение обычно намного проще чем однопоточный парсер на C

Сравнение производительности

Работа с сетью

Доступность всех необходимых системных вызовов для построения низкоуровневых библиотек

Отличная производительность среди всех динамических языков

Разница с языком C не более чем в 3 раза однако скорость разработки гораздо выше

Связка сеть+регэкспы позволяет быстро писать прототипы, которые во многих случаях можно больше не ускорять за ненадобностью

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. **Примеры проектов**
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Примеры проектов

Сайты визитки. Такие проект на языке perl сейчас реализовываются крайне редко и это понятно.

Сегодня большая часть веб-разработки — это конструктор.

- Perl занимал нишу языков для веб проектов долгое время только из-за того, что не было других альтернатив
- языки, которые создавались именно для легковесных web-проектов заняли нишу CMS
- не на всех виртуальных хостингах есть поддержка perl
- главная проблема веб-разработки на Perl заключается в том, что вы не можете скачать архив с фреймворком, распаковать, и пихать свой код в папочку src

Примеры проектов

Сайты средней нагрузки и размера

Хороший фреймворк рано или поздно встанет вам поперек работы. У него есть целых два способа сделать это: быть слишком негибким, чтобы вы возопили в попытках вписаться в него, либо же быть достаточно гибким, чтобы вы взвыли от слабой связанности и размазаности алгоритмов по коду.

Perl выигрывает у других за счёт наличия большого количества инструментов для масштабирования.

Примеры проектов

Монстры

Компании активно использующие Perl в инфраструктуре или имеющие сайты на нем:

- Mail.ru Group
- Yandex
- Amazon
- Booking
- New York Times
- DuckDuckGo
- DigitalOcean
- BuzzFeed
- Abills

Коробочные продукты на Perl:

- Bugzilla
- SpamAssassin
- twiki

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. **Документация (perldoc)**
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Документация (perldoc)

- Где можно читать про perl?

Документация (perldoc)

- Где можно читать про perl?
- Откуда брать документацию?

Документация (perldoc)

- Где можно читать про perl?
- Откуда брать документацию?
- Как изучать?

Документация (perldoc)

- Где можно читать про perl?
- Откуда брать документацию?
- Как изучать?

Самое полное и точное собрание документации находится в perldoc-e

Документация (perldoc)

Perldoc — это утилита распространяемая с дистрибутивом perl.

Она помогает:

- быстро находить необходимую информацию
- узнавать о нововведениях
- находить новые и пополнять существующие знания о правильном использовании языка и стилистике написания программ

Документация (perldoc)

С чего начать?

Документация (perldoc)

С чего начать?

Конечно же с запуска `perldoc perl`

Документация (perldoc)

Тут можно найти название разделов с обучающими материалами

perlrequick, perlretut — быстрое начало работы с регулярными выражениями и полное описание работы с ними

perlboot, perltoot, perltoot, perlbot — perl и ООП или ООП в perl

perlstyle — стилистика написания программ

perlfac* - часто задаваемые вопросы по перлу

Документация (perldoc)

Так же можно найти и техническую документацию.

perlsyn — синтаксис языка perl.

Синтаксис языка

- Программа - последовательность конструкций (statement)
- Синтаксис близок к языку C
- Perl заимствовал идеи и конструкции из C, shell, awk, sed

Синтаксис языка

Простые конструкции (**STMT**)

```
$a = 42;  
say "test";  
eval { ... };  
do { ... };  
my $var;  
# Комментарий
```

Простые конструкции возвращают значение

Синтаксис языка

Блок

```
{  
statement;  
statement;  
...  
}
```

Синтаксис языка

Управляющие конструкции

- Условия: `if`, `unless`, `elsif`, `else`
- Циклы: `while`, `until`, `for`, `foreach`
- Выбор: `given`, `when`
- Безусловный переход: `goto`

Синтаксис языка

Условия - **if**

```
if      ( EXPR ) { ... }
```

```
if      ( EXPR ) { ... }  
else    { ... }
```

```
if      ( EXPR ) { ... }  
elseif  ( EXPR ) { ... }
```

```
if      ( EXPR ) { ... }  
elseif  ( EXPR ) { ... }  
else    { ... }
```

elseif, а не **elseif** или **else if**

Синтаксис языка

Условия - `unless`

```
unless ( EXPR ) { ... } # ≡ if ( not EXPR )
```

```
unless ( EXPR ) { ... }  
else          { ... }
```

```
unless ( EXPR ) { ... }  
elsif  ( EXPR ) { ... }
```

```
unless ( EXPR ) { ... }  
elsif  ( EXPR ) { ... }  
else          { ... }
```

Циклы - `while` / `until`

```
while ( EXPR ) { ... }
```

```
while ( EXPR ) { ... } continue { ... }
```

```
until ( EXPR ) { ... }
```

```
until ( EXPR ) { ... } continue { ... }
```

Циклы - `for` / `foreach`

`for` \equiv `foreach`

Циклы - **for** / **foreach**

```
for ( EXPR; EXPR; EXPR ) { ... }
```

```
for ( LIST ) { ... }
```

```
STMT for ( LIST );
```

```
for VAR ( LIST ) { ... }
```

```
for ( LIST ) { ... } continue { ... }
```

```
for VAR ( LIST ) { ... } continue { ... }
```

Документация (perldoc)

Так же можно найти и техническую документацию.

perlsyn — синтаксис языка perl.

perldata — типы данных используемых в перле. Типы данных? В перле есть типы данных?

Переменные - типы

- SCALAR
- ARRAY
- HASH

Переменные - типы

- SCALAR
 - Number
 - String
 - Reference
- ARRAY
 - Of scalars
- HASH
 - Key: string
 - Value: scalar

Переменные - типы

- SCALAR (`$s`)
 - Number (`$s = 1`, `$s = -1e30`)
 - String (`$s = "str"`)
- Reference
 - Scalar (`$$r`, `${ $r }`)
 - Array (`@$r`, `@{ $r }`, `$r->[...]`)
 - Hash (`_%r`, `%{ $r }`, `$r->{...}`)
 - Function (`&$r`, `&{ $r }`, `$r->(...)`)
 - Filehandle (`*$r`)
 - Reference (`$$r`, `${ $r }`)
- ARRAY (`@a`, `$a[...]`)
- HASH (`%h`, `$h{key}`, `$h{...}`)

Специальные переменные

- `$" $LIST_SEPARATOR` - разделитель при интерполяции в кавычках
- `$, $OUTPUT_FIELD_SEPARATOR` - разделитель между элементами списка при выводе
- `$/ $INPUT_RECORD_SEPARATOR` - разделитель входного потока для `readline`
- `$\ $OUTPUT_RECORD_SEPARATOR` - разделитель выходного потока для `print`
- `$. $INPUT_LINE_NUMBER`

Специальные переменные

```
$" = "."; # $LIST_SEPARATOR
$, = ";"; # $OUTPUT_FIELD_SEPARATOR
$\ = "\n\n"; # $OUTPUT_RECORD_SEPARATOR
while (<>) {
    chomp;
    @a = split /\s+/, $_;
    say "$. @a",@a;
}
```

Документация (perldoc)

Так же можно найти и техническую документацию.

perlsyn — синтаксис языка perl.

perldata — типы данных используемых в перле. Типы данных? В перле есть типы данных? Да там есть 3 типа данных «Скаляр» \$var, «Массив» @var, «Ассоциативный массив» (хэш) %var.

perlvar - специальные переменные

perlop — операции которые поддерживает перл.

perlsub - всё что касается объявления и использования подпрограмм.

perlfunc - описание встроенных функций

perlpod* - документирование кода

perlmod* - модульность в перле

perlxs*, perlembed, perlcalls - интеграция языка perl и C

Документация (perldoc)

Это набор наверно основных точек документации с которыми вам придётся сталкиваться на протяжении первых лекций и при выполнении домашних заданий.

Но это далеко не все разделы документации которые есть в perldoc

На протяжении всего курса мы будем указывать ссылки на документацию

Так же документацию можно смотреть на <http://perldoc.perl.org>

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. **Настройка окружения**
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Настройка окружения

perl завоевал к себе доверие, за счет того, что был портирован под всевозможные платформы и системы

ALL	cygwin	darwin	dragonfly	freebsd	gnu	gnukfreebsd	linux	midnightbsd	mirbsd	mswin32	netbsd	openbsd	solaris
5.21.9													
5.21.8													
5.21.7													
5.21.6													
5.21.5													
5.21.4													
5.21.3													
5.21.2													
5.21.1													
5.21.0													
5.20.2													
5.20.1													
5.20.0													
5.19.12													
5.19.11													
5.19.10													
5.19.9													
5.19.8													
5.19.7													
5.19.6													
5.19.5													
5.19.4													
5.19.3													
5.19.2													
5.19.1													
5.19.0													
5.18.4													
5.18.3													
5.18.2													
5.18.1													
5.18.0													

Настройка окружения

MacOSx сам перл уже предустановлен

А вот для сборки XS-модулей, вам придётся установить себе xcode + “Command line tools”

Подробнее про установку можно прочитать в `perldoc perlmacosx`

Так же возможно вам может понадобится `macports`

Настройка окружения

Linux.

Как правило дистрибутив языка перл есть в репозитории системы портов.

Например:

- для CentOS установка выполняется посредством вызова `yum install perl`
- для Debian: `apt-get install perl`
- для Gentoo: `emerge dev-lang/perl`
- для FreeBSD: `pkg install perl5`

Для любителей компилировать вёс под свою систему:

- `make perl ./Configure -des; make test instal`

Настройка окружения

Windows.

Тут выбор велик, я отмечу самые популярные из вариантов:

- ActivePerl от ActiveState
- StrawberryPerl
- cygwin

Мы рекомендуем использовать StrawberryPerl

В отличие от ActivePerl он идёт сразу с компилятором mingw и установка модулей в нём становится гораздо удобнее и проще

С ActivePerl надо позаботиться о наличии nmake + win32GnuUtils иначе сборка модулей будет для вас мучительной и утомляющей

Настройка окружения

После установки проверяем доступность интерпретатора: `perl -v`

Выдача будет примерно такой

```
This is perl 5, version 18, subversion 2 (v5.18.2)
x86_64-linux-gnu-thread-multi
(with 40 registered patches, see perl -V for more details)

Copyright 1987-2013, Larry Wall

Perl may be copied only under the terms of either the
GNU General Public License, which may be found in the
documentation, or the Perl Artistic License, which may be found in the
documentation.

Complete documentation for Perl, including FAQ list, is
available on-line and you should read this system using "man perl" or "perldoc perl". If
you have access to the Internet, point your browser at http://www.perl.org
```

Настройка окружения

Если вы видите, что то в виде «Command not found». Или другие ошибки, то с установкой, что то пошло не так.

Возможно интерпретатор находится за пределами все возможных путей из переменной окружения среды PATH

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. **Запуск скриптов (perlrun)**
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Запуск скриптов (perlrun)

Разберёмся с запуском и выполнением простейших однострочных скриптов.

Именно с этого и пошло развитие этого языка

Самое приятное — это то, что perl под всеми системами работает одинаково, за исключением системнозависимых библиотек.

Запуск скриптов (perlrun)

Ключ -e сообщает интерпретатору, что следующую за ключом строку надо выполнить, как скрипт:

```
perl -e 'print "Hello world\n"'
```

Однако будьте аккуратны с кавычками, не все шеллы одинаково с ними работают.

Запуск скриптов (perlrun)

Обычно однострочная программа, обрабатывает данные приходящие ей в стандартном вводе.

Вот простейшая программа которая читает стандартный ввод и распечатывает строки добавляя в начало каждой из них дефис

```
perl -e 'while(<>){print "- ".$_}'
```

Запуск скриптов (perlrun)

Обычно однострочная программа, обрабатывает данные приходящие ей в стандартном вводе.

Вот простейшая программа которая читает стандартный ввод и распечатывает строки добавляя в начало каждой из них дефис

```
perl -e 'while(<>){print "- ".$_}'
```

Но кто захочет писать каждый раз такую конструкцию?

```
while(<>){}
```

Запуск скриптов (perlrun)

Обычно однострочная программа, обрабатывает данные приходящие ей в стандартном вводе.

Вот простейшая программа которая читает стандартный ввод и распечатывает строки добавляя в начало каждой из них дефис

```
cat 'file_name.txt' | perl -e 'while(<>){print "- "
```

Но кто захочет писать каждый раз такую конст рукцию?

```
while(<>){}
```

Для упрощения написания таких однострочников был придуман ключ -n

```
cat 'file_name.txt' | perl -ne 'print "- ".$_'
```

Запуск скриптов (perlrun)

Внутри вот такого цикла:

```
while(<>){}
```

Переменная `$_` будет содержать целную строку вместе с символом `\n`

Что бы не использовать команду `chomp` (отрезающую в конце перенос строки) можно использовать флаг `-l` :

- устанавливает переменную `$\` (разделитель, который выведен после каждого выполнения команды `print`)
- устанавливает переменную `$/` (разделитель по которому будет делится входящий поток, отдельно его можно выставить с помощью флага `-0`)
- удаляет из строки `$_` последний перенос строки (при совместном использовании с флагом `-n`)

Запуск скриптов (perlrun)

Флаг -р делает тоже самое, что и флаг -n, только в каждую итерацию цикла добавит еще вывод переменной \$_

Таким образом прошлый пример можно преобразовать:

```
perl -pl00120073 -e ' '
```

```
perl -pl00730012 -e ' '
```

Очень похоже на программу, которая сама за нас пишет программы)))

Детальнее можно посмотреть тут: `perldoc perlvar`, `perldoc perlrun`

Запуск скриптов (perlrun)

Для парсинга более сложных структур файлов, например когда в каждой строке есть записи разделенные определённым разделителем, можно использовать флаг -a совместно с -F

-a добавляет функцию разделяющую входную строку на части и складывает в спецмассив @F

-F устанавливает разделитель, по умолчанию это пробел

Например вам надо прочитать файл-таблицу, в которой каждая строка представляет собой набор полей разделенных ";", проверить третью колонку на наличие там 1 и при выполнении условия вывести значение из 2 колонки

```
perl -lnaF';' -e 'if( $F[2] == 1 ){ print $F[1] };'
```

!О переводах строк за нас побеспокоился флаг -l

Запуск скриптов (perlrun)

Если ваша программа требует подключения модулей, то подключить модули можно опцией -M, например:

```
perl -MJSON::XS -e 'print JSON::XS::encode_json({va
```

Запуск скриптов (perlrun)

Это не все ключи которые поддерживает перл, но их должно хватить, для начала изучения.

А теперь немного вернёмся к истории. Я уже говорил о том, что perl 5 версии был переосмыслен и почти полностью переписан. Так вот начиная с версии 5.005 был сделан доступ к компилятору.

Теперь мы можем заглянуть во внутренности компилятора perl. Посмотреть как будет выглядеть ваш код, посмотреть на информацию компиляции.

Запуск скриптов (perlrun)

Для таких модулей был выделен неймспейс B::

А доступ к компилятору реализован в модуле "O"

```
perl -MO=Backend
```

Вот такой нехитрой записью можно определить модуль которому будет передана ваша программа после компиляции.

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. **Модуль Deparse**
9. Модуль Data::Dumper
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Модуль Deparse

B::Deparse - очень полезный модуль, который способен превратить опкоды вышедшие после компилятора в пелркод.

Другими словами декомпилировать opcode обратно в perl

B::Concise - бекенд позволяющий посмотреть опкоды как они есть

Возьмём уже известный нам пример:

```
perl -pl00730012 -e ''
```

И попробуем воспользоваться модулем O передав ему параметром модуль B::Deparse

```
perl -M0=Deparse -pl00730012 -e ''
```

Модуль Deparse

На выходе получим

```
BEGIN { $/ = "\n"; $\ = ";"; }  
LINE: while (defined($_ = <argv>)) {  
    chomp $_;  
}  
continue {  
    die "-p destination: $_\n" unless print $_;  
}  
-e syntax OK
```

Модуль Deparse

B::Deparse можно использовать, как обычный модуль

Если вам необходимо посмотреть на код функции по ссылке на эту функцию

```
use B::Deparse;
sub func {
    print 'Hello world!!!'
};

my $deparse = B::Deparse->new("-p", "-sC");
$body = $deparse->coderef2text(\&func);

print $body;
```

Выполнив такую программу, на стандартный вывод мы получим:

```
{
    print('Hello world!!!');
}
```


Промежуточные итоги

Итого на данном этапе вы уже знаете

- как установить perl
- где искать документацию
- как запускать простейшие однострочники
- как изучать поведение компилятора с вашими однострочниками

Далее мы перейдём к модулю который помогает разворачивать сложные структуры данных

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. **Модуль Data::Dumper**
10. Модуль DDP
11. Отладка perl скриптов (perldebug)

Модуль Data::Dumper

Data::Dumper - модуль, который поможет выводить на экран в развернутом виде сложные структуры данных.

```
use Data::Dumper;  
my $foo = [{a => 1, b => 2},{c => 3, d => 4}];  
print Dumper($foo);
```

Вот так красиво демонстрирует эту переменную Data::Dumper

```
$VAR1 = [  
  {  
    'b' => 2,  
    'a' => 1  
  },  
  {  
    'c' => 3,  
    'd' => 4  
  }  
];
```

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. **Модуль DDP**
11. Отладка perl скриптов (perldebug)

Модуль DDP (Data::Printer)

Есть альтернативный модуль для просмотра структур и объектов. Его обычно используют для дебага приложений.

У этого модуля не меньше настроек, чем у Data::Dumper, но его проще использовать, например:

```
use DDP;  
my $foo = {a=>1, b=> 2, c=> [1,2,3]};  
p $foo;
```

На выходе:

```
\ {  
  a  1,  
  b  2,  
  c  [  
    [0] 1,  
    [1] 2,  
    [2] 3  
  ]  
}
```

Модуль DDP (Data::Printer)

Некоторые отличия от Data::Dumper

Автор этого модуля позаботился о цветовой разметке выводимых данных, для удобства чтения.

Так же у этого модуля более расширенный дамп объектов, но нет сериализации

```
\ SomeClass {
  Parents      Moose::Object
  Linear @ISA   SomeClass, Moose::Object
  public methods (3) : bar, foo, meta
  private methods (0)
  internals: {
    _something => 42,
  }
}
```

Содержание

1. Цель курса
2. История создания языка Perl
3. Сравнение производительности
4. Примеры проектов
5. Документация (perldoc)
6. Настройка окружения
7. Запуск скриптов (perlrun)
8. Модуль Deparse
9. Модуль Data::Dumper
10. Модуль DDP
11. **Отладка perl скриптов (perldebug)**

Отладка perl скриптов (perldebug)

Для начала работы с дебагером рекомендую прочитать документацию perldebtut

Запуск отладчика выполняется добавлением ключа -d при запуске интерпретатора

```
perl -d myscript.pl
```

Для того, что бы отладчик запустился скрипт не должен содержать синтаксических ошибок и должен нормально компилироваться perl -c

Отладка perl скриптов (perldebug)

После запуска отладчика вы увидите на экране

```
Loading DB routines from perl5db.pl version 1.44
Editor support available.

Enter h or 'h h' for help, or 'perldoc perldebug' for
DB<1>
```

#

Далее отладчик ждёт от вас команд

Отладка perl скриптов (perldebug)

Отладка программ подразумевает построчное их выполнение с возможностью просмотра состояния переменных между ними.

Одним из вариантов отладки является вывод в STDERR состояние переменных в том или ином месте кода.

Каждый из вариантов является по своему удобный.

Отладка perl скриптов (perldebug)

Как это выглядит. Скрипт:

```
use strict;

my $ret = 0;

foreach(my $i = 0; $i < 50; $i++){
    if($ret>$i){
        $ret -= $i;
    }
    else {
        $ret += $i;
    }
}
print $ret;
```

Отладка perl скриптов (perldebug)

Запускаем

```
perl -d mydebug.pl
```

Отладчик запустился и выдал приглашение

```
Loading DB routines from perl5db.pl version 1.44  
Editor support available.
```

```
Enter h or 'h h' for help, or 'perldoc perldebug' for
```

```
main::(mydebug.pl:3):    my $ret = 0;
```

```
DB< 1 >
```

Отладка perl скриптов (perldebug)

Просмотр кода с текущей позиции командой l

```
3==>    my $ret = 0;
4:       my $cnt_add = 0;
5:       my $cnt_sub = 0;
6
7:       foreach(my $i = 0; $i < 50; $i++){
8:           if($ret>$i){
9:               $ret -= $i;
10          }
11          else {
12:              $ret += $i;
```

Установим точку останова на 8 строку при достижении 10 итерации цикла:

```
b 8 $i == 9
```

Запустим программу

c

Отладка perl скриптов (perldebug)

Когда выполнится условие `$i == 9` сработает точка останова и на экране появится сообщение

```
DB< 7 >
```

```
c
```

```
main::(mydebug.pl:8):                                if($ret > $i){
```

Выведем значение переменной `$i`

```
DB< 7 > print $i
```

```
9
```

А так же поставим отслеживать переменную `$ret`:

```
w $ret
```

Отладка perl скриптов (perldebug)

Выполним следующую строку скрипта:

```
DB< 9 >
```

```
n
```

```
Watchpoint 0:  $ret changed:
```

```
old value:  ''
```

```
new value:  '16'
```

```
main::(mydebug.pl:9):
```

```
$ret -= $i;
```

```
DB< 9 >
```

Дальнейшая отладка происходит по тем же принципам

Домашнее задание

1. Написать однострочную программу на perl, которая на вход получает список файлов выводимых командой `ls -l` или `dir`, полученный поток разбивает на колонки и выводит в виде таблицы с разделителем ';'. (Важно, имена файлов не должны содержать символа разделителя, но могут содержать пробелы). Результат работы программы средствами шела сохраняем в файл.
2. Написать однострочную программу, которая прочитает со стандартного входа файл созданный в 1 задании. И выведет на экран название файлов размер которых больше 1 мегабайта. Последней строкой надо вывести общее кол-во строк в таблице, и количество строк удовлетворяющее условию.
3. Написать программу которая возьмёт файл из задания 1 и построит структуру данных в виде массива массивов, после чего распечатает его при помощи модулей `DDP` и `Data::Dumper`
4. Провести отладку скрипта написанного в задании 3. Создать файл сценарий для отладчика.



Оставьте отзыв

Спасибо за внимание!

Николай Шуляковский

Email & Agent: n.shulyakovskiy@corp.mail.ru