# Скриптовые языки программирования

Доклад на семинаре по специальности

Студент гр. 4057/2  Руцкий Владимир

20.10.2009

# Содержание

- Введение
- Особенности скриптовых языков программирования
- Типы скриптов и примеры
- Заключение

# Введение

# Иерархия языков программирования

- Машинные коды, язык ассемблера. Работа с отдельными ячейками памяти, регистрами
- Компилируемые в машинные коды (C, C++). Работа с примитивными типами данных (числа, массивы, структуры/классы)
- Компилируемые в байт-код или интерпретируемые (Java, C#, Python). Более сложные типы данных и более сложные элементарные операции над ними
- Работающие со сложными типами данных (с таблицами, программами — SQL, shell)

# Сложность элементарных операций

<здесь будет приведена диаграмма количества машинных команд необходимых для выполнения каждой элементарной операции в соответствующем классе языка программирования с предыдущего слайда>

# Сложность написания программ

<здесь будет приведена статистика, сколько строк кода (и сколько времени) занимает решение какой-то определённой задачи, в различных классах языков программирования с предпредыдущего слайда>

# Эволюция языков программирования

Чем выше уровень языка, тем

- Проще и быстрее разработка программы
- Медленнее работа программы

Вычислительная техника развивается столь стремительно, что применение сильно высокоуровневых языков программирования является оправданным во многих задачах.

# Появление скриптов

- 1960-е годы. Shell-скрипты. Автоматизация работы человека-оператора по вводу команд ОС в системах с разделением времени
- «Скриптовый язык», «язык сценариев» - язык записи «сценария», последовательности выполнения команд ОС

# Особенности скриптовых языков программирования

# Назначение скриптовых языков программирования

Быстрое и простое связывание и управление готовыми объектами (функциями, программами)

- Для создания новых программ на основе существующих
- Для автоматизации различных рутинных операций
- Для быстрой разработки технологических тестов
- Для задания сценариев работы программы не программистами
- Для управления специальными данными

# Типы данных

- Бестиповые — более абстрактный и универсальный код
- Универсальные типы — возможность произвольного связывания различных компонент

Недостаток:

- Нетипизированность данных не позволяет выявить ошибочное использование переменных до начала выполнения скрипта

# Особенности среды выполнения скриптов

Выполнение скриптов:

- Практически никогда не компилируются в машинные коды
- Компилируются в байт-код (обычно лишь для оптимизации скорости выполнения, кешируются)
- Чаще всего интерпретируются «на лету»

Скрипты часто являются кроссплатформенными: для переноса скриптов на новую платформу достаточно перенести на неё среду выполнения скриптов (интерпретатор/компилятор в байт код) и библиотеки.

# Особенности синтаксиса

- Чаще всего позволяет «построчное» выполнение  кода — даёт возможность программирования «на лету», прямо во время выполнения программы
- Минимализм в конструкциях языка

# Типы скриптов и примеры

# Управление последовательностью команд ОС

- «Shell» — «оболочка» — интерфейс между функциями ядра/системы и конечным пользователем
  - GUI — graphical user interface
  - CLI — command-line interface
- Первые оболочки ОС — текстовые (CLI)

# История Shell

- CTSS OS (Compatible Time-Sharing System) — одна из первых ОС с разделением времени (1961, MIT Computation Center)
- Louis Pouzin (родился в 1931 во Франции) участвовал в проектировании CTSS, разработал программу RUNCOM (1963/64), позволяющую выполнять команды ОС в текущей директории. Первым ввёл термин «shell»

# Развитие Shell

<TODO: пока не знаю точно о чем это>

Четыре поколения shell:

1. Thomson shell, Mashey shell
2. C-shell, Bourne shell
3. tcsh, ksh88
4. ksh93, bash, zsh, Microsoft Power Shell

# ОС Unix shell: sh

<TODO: Изменить заголовок слайда, мне не удаётся сделать это в OpenOffice сейчас :)>

- **Thompson shell** — первая оболочка для Unix, разработал Ken Thompson в в AT&T Bell Laboratories в 1971
- **sh** — *Bourne shell*, разработал Stephen Bourne в AT&T Bell Laboratories и был выпущен в 1977 как оболочка по умолчанию для Version 7 Unix. Заменил Thompson shell
- **bash** — *Bourne-again shell, р*азработал в 1987 Brian Fox в рамках проекта GNU.
- bash — обратно совместимое надмножество над sh
- bash до сих пор продолжает развиваться и широко используется в качестве shell по умолчанию в различных Unix-like ОС

# Пример работы с Unix-системой через CLI - bash (1/2)

```
bob@bob:~$ slogin balto -l bob
bob@balto's password:
Linux balto 2.6.24.2 #5 SMP Sat Jul 26 18:43:21 MSD 2008 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
Last login: Fri Oct 16 03:14:01 2009 from bob
bob@balto:~$ ls
backup  Desktop  stuff
bob@balto:~$ ls -l
total 12
drwx------ 2 bob bob 4096 2008-08-22 01:50 backup
drwx------ 2 bob bob 4096 2008-12-23 21:51 Desktop
drwxr-xr-x 6 bob bob 4096 2009-10-16 03:12 stuff
bob@balto:~$ cat > hello.cpp
#include <iostream>

int main()
{
  std::cout << "Hello, world!\n";
}
```

# Пример работы с Unix-системой через CLI - bash (2/2)

```
bob@balto:~$ g++ -Wall -o hello hello.cpp
bob@balto:~$ ls -l
total 28
drwx------ 2 bob bob  4096 2008-08-22 01:50 backup
drwx------ 2 bob bob  4096 2008-12-23 21:51 Desktop
-rwxr-xr-x 1 bob bob 10304 2009-10-16 03:16 hello
-rw-r--r-- 1 bob bob    71 2009-10-16 03:15 hello.cpp
drwxr-xr-x 6 bob bob  4096 2009-10-16 03:12 stuff
bob@balto:~$ ./hello
Hello, world!
bob@balto:~$ cat hello.cpp
#include <iostream>

int main()
{
  std::cout << "Hello, world!\n";
}

bob@balto:~$ rm hello
bob@balto:~$ ls -l
total 16
drwx------ 2 bob bob 4096 2008-08-22 01:50 backup
drwx------ 2 bob bob 4096 2008-12-23 21:51 Desktop
-rw-r--r-- 1 bob bob   71 2009-10-16 03:15 hello.cpp
drwxr-xr-x 6 bob bob 4096 2009-10-16 03:12 stuff
bob@balto:~$ logout
Connection to balto closed.
bob@bob:~$
```

# Возможности bash

- Сейчас bash — полноценный язык программирования, по историческим причинам ориентированный на задание последовательности выполнения команд ОС
- «Стандартной библиотекой» для bash является набор стандартных утилит Unix.

# Возможности bash, как языка программирования

<переменные, конструкции if-then-else, for, while, switch, возможность задания функций, массивы, перенаправление ввода/вывода. Примеры>

# Возможности bash в связке со стандартным набором программ Unix

<find, grep и некоторые примеры>

# ОС MS Windows CLI: cmd.exe

- **COMMAND.COM** — shell по умолчанию для ОС DOS, и CLI ОС MS Windows 9x/Me <TODO: где и когда был разработан впервые>
- **CMD.EXE** — shell для ОС MS Windows выше Windows 2000, разработал Therese Stowell для MS Windows NT

# Возможности cmd.exe

<TODO>

<+ Примеры>

# Другие оболочки ОС

- <Другие Unix shells>
- <MS Power Shell>

# Итог: современное использование оболочек ОС

<TODO>

# Языки для обработки текстов

- Текст — цепочка символов — универсальный тип данных
- Алгоритмы, принимающие цепочку символов и возвращающие цепочку, можно связывать друг с другом в произвольном порядке
- Shell превосходно подходит для обработки цепочек символов: вход и выход программ — цепочки символов и shell имеет встроенную возможность для перенаправления вывода одной команды ОС на вход другой

# sed

- <История создания, возможности, примеры>

# awk

- <История создания, возможности, примеры>

# Web-скрипты

- Чаще всего обмен информацией организован внутри пар клиент-сервер
- Общие для большинства схем взаимодействия клиент-сервер рутинные операции:
  - генерация данных на сервере,
  - подготовка к передаче клиенту,
  - отображение данных на стороне клиента,
  - приём данных от клиента

# Серверные Web-скрипты

- <PHP, ASP, Ruby on rails, SMX>

# Web-скрипты. Клиентские

- <JavaScript, VBScript>

# Языки общего назначения

# Perl. Описание

- **Perl** — Practical Extraction and Reporting Language — высокоуровневый язык общего назначения.
- Первоначально разработал Larry Wall в 1987 в США для обработки отчетов
- Предоставляет широкие возможности для обработки текстов
- Практичный язык программирования
- Процедурный, с поддержкой объектно-ориентированной и функциональной парадигм программирования
- Автоматическое управление памятью. Сборщик мусора
- Интерпретируемый. В Perl 6 появится возможность компилирования в байт-код
- Девизы:
  - «There's more than one way to do it» (TMTOWTDI) — «Есть больше одного способа сделать это»
  - «Easy things should be easy and hard things should be possible» — «Простые вещи должны быть простыми, а сложные вещи — возможными»

# Perl. Синтаксис

- Унаследовал общую структуру от языка Си:
  - процедурный,
  - имеет переменные, выражения, операторы присваивания,
  - код разделяется фигурными скобками,
  - имеет управляющие структуры и возможность написания функций,
  - поддержка разделения на модули
  - поддержка ссылок

```
sub GCD {
  my ($a, $b) = @_;
  my $c = $b;
  while ($a > 0)
  {
    $c = $a;
    $a = $b % $a;
    $b = $c;
  }
  return $c;
}
print GCD(42, 56);
```

# Perl. Типы данных

- Базовые типы данных:
  - скаляры (числа, строки и ссылки),
  - массивы,
  - хеши,
  - потоки ввода/вывода,
  - и некоторые другие типы
- Сложные типы данных реализуются через массивы/хеши ссылок на более простые типы
- Примеры:

```
$number = 5;
$string = "String";
$multilined_string = <<EOF;
Multilined string,
terminating with the word "EOF".
EOF
```

```
@array = (1, 2, "three");
print $array[1]; # "2"

%hash = ('one' => 1, 'two' => 2);
print $hash{'one'}; # "1"

$ref = \$number;
print $ref; # SCALAR(0x14ef640)
print $$ref; # 5

$circle={
  center=>{x=>210, y=>297},
  radius=>53,
};
```

# Perl. Конструкции языка

- Perl богат «синтаксическим сахаром»
- Некоторые конструкции языка:

```
while ( cond ) { ... } [continue { ... }]
for ( init-expr ; cond-expr ; incr-expr ) { ... }
foreach var ( list ) { ... } [continue { ... }]
if ( cond ) { ... } [[elsif ( cond ) { ... }] else { ... }]
given ( expr ) { when ( cond ) { ... } default { ... } }
```

```
statement if cond ;
statement unless cond ;
statement while cond ;
statement until cond ;
statement foreach list ;
```

```
expr1 and expr2
expr1 && expr2
expr1 or expr2
expr1 || expr2
```

# Perl. ООП

- Пример. Поддержка ООП:

```
package Animal;
sub new {
  my $class=shift;
  my $self={name=>shift, sound=>shift};
  return bless $self, $class;
}

sub whoAmI {
  my $self=shift;
  print "Здравствуйте, я $self->{name}. ";
}

sub say {
  my $self=shift;
  print "$self->{sound}!\n";
}
```

```
package main;
```

- my $cow=new Animal 'Корова', 'Муу';
- my $cat=new Animal 'Кошка', 'Мяу';
- my $dog=new Animal 'Собака', 'Гав';
- $cow->whoAmI; $cow->say;
- $cat->whoAmI; $cat->say;
- $dog->whoAmI; $dog->say;

- Вывод скрипта:

```
Здравствуйте, я Корова. Муу!
Здравствуйте, я Кошка. Мяу!
Здравствуйте, я Собака. Гав!
```

# Perl. Регулярные выражения

- Поддержка регулярных выражений:
  - $x =~ m/abc/; # Истина, если $x содержит «abc»
  - $x =~ s/abc/def/; # Заменит «abc» на «def» в $x
  - $x =~ m/(\d+)\.(\d+)\.(\d+)\.(\d+)\./ # Найдёт первое вхождение IP-адреса, и сохранит его части в переменных $1, $2, $3, $4
  - @a = ($x =~ m/(\d+)/g); # Сохранит в массив @a все найденные в строке числа
  - $x =~ s/(\d+) б /$1 Б/g # Заменит «X б» на «X Б»
- Регулярные выражения Perl удобны и включены во многие другие языки программирования
- Пример. Печать простых чисел:

```
perl -wle '(1 x $_) !~ /^(11+)\1+$/ && print while ++ $_'
```

# Perl. Работа с потоками

# Perl. Функциональное программирование

# Perl. Применение (1/2)

- Применение Perl:
  - Разработка инструментов системного администрирования
  - Обработка почты
  - CGI-сценарии
  - Обработка текстов

# Perl. Применение (2/2)

- Продукты, использующие Perl:
  -
- Компании, использующие Perl:
  -

# Python. Описание



- **Python** — высокоуровневый язык общего назначения
- Первоначально разработал Guido van Rossum в 1990 в нидерландском CWI (нем. Centrum voor Wiskunde en Informatica — National Research Institute for Mathematics and Computer Science)
- Акцент на производительность разработчика  и читаемость кода
- Объектно-ориентированный, с поддержкой процедурной и функциональной парадигм
- Минималистичный синтаксис
- Динамическая типизация
- Автоматическое управление памятью. Сборщик мусора
- Интерпретируемый и/или компилируемый в байт-код
- Полная интроспекция
- Механизм обработки исключений
- Интерактивный режим выполнения

# Python. Синтаксис



- Имеет переменные, выражения, операторы присваивания
- Отступы определяют разделение программы на блоки
- Имеет управляющие структуры, возможность написания функций и классов
- Поддержка разделения на модули
- Переменные — имена ссылок на значения

```
def GCD(a, b):
    while b > 0:
        a,b = b,a%b
    return a

print GCD(42, 56)
```

# Python. Типы данных (1/2)


- Базовые типы:
  - Логический тип
  - Числа
    - целые,
    - с плавающей точкой,
    - неограниченной точности,
    - комплексные
  - Строки (с поддержкой Unicode)
  - Массивы байт

- Передаются по значению

# Python. Типы данных (2/2)


- Коллекции (контейнеры):
  - Списки
  - Кортежи
  - Словари
  - Множества

  - Передаются по ссылке
- Примеры:

```
number = 5
complex_number = 1.5 + 0.5j
print complex_number.imag # "0.5"
big_number = 2**1000
print len(str(big_number)) # "302"

string = "String"
multilined_string = """
Multilined string,
yep.
"""
unicode_string = u"Уникод"

list_var = [1, 2, 'three']
tuple_var = (1, 2, 'three')
print list_var[1] # "2"

dictionary = {'one': 1, 'two': 2}
print dictionary['one'] # "1"
set_var = set([1, 2, 'five'])
print 'five' in set_var # "True"
```

# Python. Конструкции языка


- Операторы:
  - **if** cond1:

    expr1

    **elif** cond2:

    expr2

    **else**:

    expr3
  - **while** cond:

    expr (возможны break, continue)
  - **class** name(parent classes):

    expr
- **def** name(args):

  expr (возможны return, yield)
- **try**:

  expr1

  **except** exception:

  expr2

  **else**:

  expr3

  **finally**:

  expr4

# Python. ООП

# Python. Функциональное программирование



- Лямбда функции
- List comprehesion

# Python. Особенности



- Итераторы и генераторы
- Интроспекция

# Python. Применение (1/2)



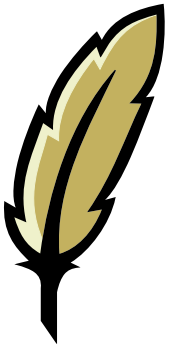
- Часто используется как основной язык программирования
- Быстрое прототипирование
- Разработка инструментов для системного администрирования
- Веб-программирование
- Встроенный скриптовый язык

# Python. Применение (2/2)



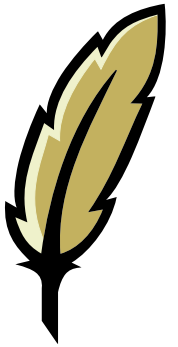
- Продукты, использующие Python:
  -
- Компании, использующие Python:
  -

# Tcl. Описание



- **Tcl** — *Tool Command Language* — «тикль» — высокоуровневый язык общего назначения
- Первоначально разработал John Ousterhout в 1988 в Калифорнийском институте в Беркли, США
- Процедурный язык программирования с поддержкой
  - метапрограммирования,
  - функциональной парадигмы,
  - модели управления программой на основе событий
- Базовая реализация не содержит поддержки ООП, но есть альтернативные реализации, включающие её
- Ортогональность
- Компилируется в байт-код, а затем выполняется

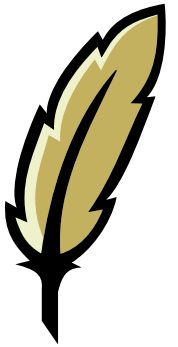
# Tcl. Синтаксис



- Скрипт состоит из команд, разделённых переводом строки или точкой с запятой
- Команда:

  CommandName arg1 arg2 … argN
- Специальные символы:
  - $ — подстановка значения переменной
  - [] — подстановка результата выполнения команды внутри скобок.
  - "" — группировка аргументов в один с подстановкой значений переменных
  - {} — группировка аргументов в один без подстановки значений переменных.

# Tcl. Конструкции языка



- Реализованы через общий синтаксис
- **set** a 5 # a = 5
- **expr** expression
  - **set** sum [**expr** $a + $b]
- **while** { cond } { expr }
  - **while** { $x < 10 } { **puts** $x; **incr** $x }
- **if** { cond } { expr }
- **proc** name { args } {

  expr (**return** expr)

  }

- **switch** options key {

  pattern1 { expr }

  pattern2 { expr }

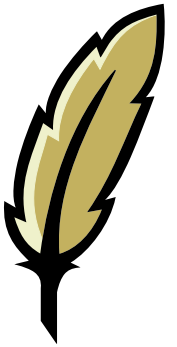
  …

  default { expr }

  }

- **foreach** iter values_list { expr }

# Tcl. Типы данных

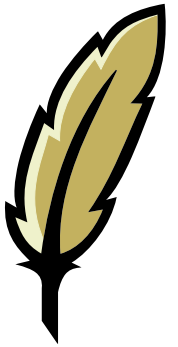


- Строки
- Списки строк
- Ассоциативный массив
- Примеры:

```
set number 5
set string "Some string with number $number"
set string_list {e1 e2 e3}
set map_var(one) 1
set map_var(two) 2
puts $map_var(two) # «2»
```

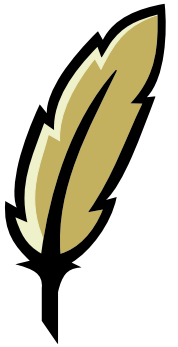
```
proc gcd { a b } {
  while { $b > 0 } {
    set t $b
    set b [expr $a % $b]
    set a $t
  }
  return $a
}

puts [gcd 42 56]
```

# Tcl. Функциональное программирование



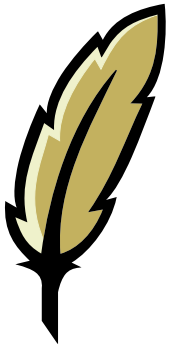
- Функциональная парадигма используется редко
- Возможность создания функций высшего порядка и абстракции функции

# Tcl/Tk. Задание GUI (1/2)



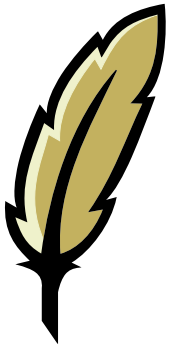
- Tk — Tool Kit — «инструментарий» — кроссплатформенная библиотека базовых элементов GUI
- Разработал John Ousterhout как расширение Tcl
- С использованием специальных библиотек может использоваться с другими языками программирования
- Представляет собой набор Tcl-команд для задания иерархии элементов GUI, и привязывания функций обработчиков к возникающим событиям

# Tcl/Tk. Задание GUI (2/2)



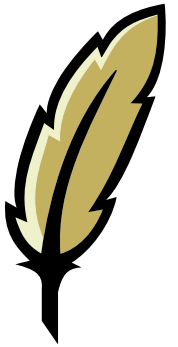
- Пример

# Tcl. Применение (1/2)



- Быстрое прототипирование
- Создание графических интерфейсов (Tcl/Tk)
- Встроенный скриптовый язык
- Тестирование
- Написание хранимых процедур PostgreSQL (PL/Tcl)
- Иногда создание CGI-скриптов

# Tcl. Применение (2/2)



- Продукты, использующие Tcl:
  -
- Компании, использующие Tcl:
  -

# Ruby

# Lua

# BASIC

# Узконаправленные языки

- <MAXScript (3DSMax), MATLAB, R, Bison, консоль в некоторых играх (Quake, Half-Life), макросы MS Word/Excel, OpenOffice.>

# Сравнение скриптовых языков программирования

<здесь будет сводная таблица характеристик различных скриптовых языков программирования, как Вы предложили>

# Заключение

Скриптовые языки — набор удобных и надёжных инструментов для быстрого и простого решения широкого спектра задач

# Источники информации

<будет заполнено позже>

# Благодарю за внимание!