ECMAScript — ассемблер будущего, бэкенд, фронтенд и все-все-все

(Об эволюции и фичах JavaScript)

Виталий Филиппов, CUSTIS

О своих предпочтениях

CUSTIS®

«И давно вы занимаетесь программизмом?»

- Начинал лет в 11 с C/C++ (Turbo C / C++Builder)
- Потом открыл для себя Linux, свободный софт…

  главное читать логи :)
- …LAMP (Perl/PHP), HTML и JS
- Теперь полюбил серверный JS (nodejs)

О чём доклад?

CUSTIS®

- Почему JS?
- История JavaScript
- Обзор языка, производительность
- Обзор выдумок

  (фреймворки, системы сборки и т.п)
- Немного демо

Что такое JS?

CUSTIS®

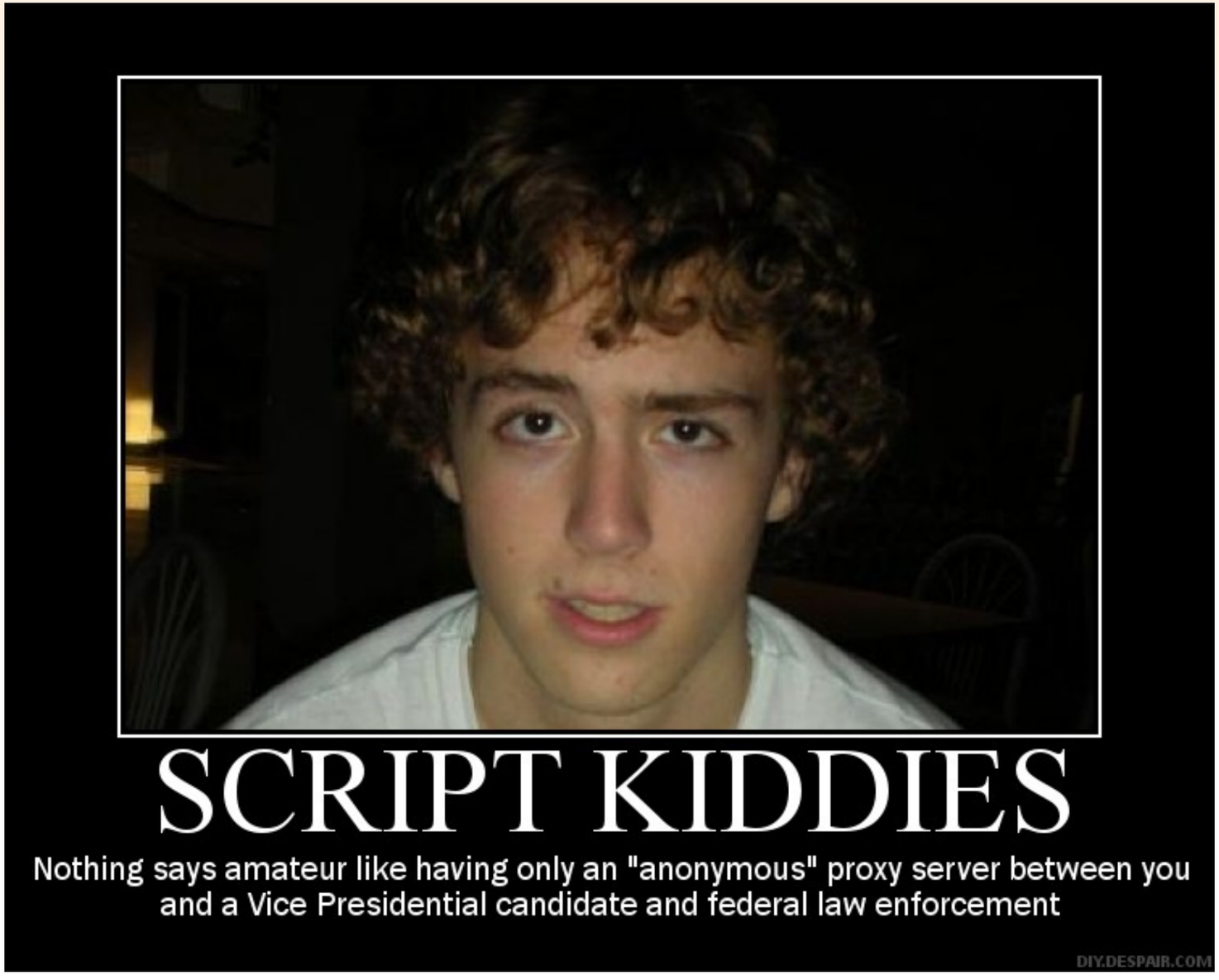
~~JavaScript~~ECMAScript

- **От Java только часть названия**
- Скриптота! (динамический язык)
- Объект/массив/скаляр (JSON)
- Прототипы, замыкания, колбэки, нет многопоточности
- ES — язык. А ещё есть окружение (DOM, BOM)
- Куча новых фич (ES2015-2016-2017)
- Браузерный — Chrome (V8), Firefox (SpiderMonkey) и даже IE (ChakraCore)
- Серверный — node.js (V8)

`ECMAScript и все-все-все`

Скриптота vs типизация

CUSTIS®



Холивар!!!

CUSTIS®

- Шутки в сторону: тема серьёзная
- Популярных динамических языков много

  можно пытаться это отрицать, но таки удобно

- Статические… хде они? Java? (C# не в счёт, винда)
- Компилируемые новые есть: D, Rust, Go, Vala, Swift

  но кто на них пишет-то?

Все хотят одного

CUSTIS®

- Типизация — не необходимость, как раньше, а лишь один из способов проверки
  - что ещё можно проверять?
  - Rust: borrow checker
  - функциональщина: «purity» checker
  - Java: checked exceptions
- При этом
  - auto уже даже в C++
  - тайпчекер (частично) уже даже в PHP (+ Hack)

Проверки — хорошо, но

Почему JS?

CUSTIS®

- Нейтральный C-подобный синтаксис
- Быстрый! (как вычислительно, так и для I/O)

  Событийная машина node.js — не пустой звук! (1M соединений на одном сервере)
- Всегда нужен в браузерах (а веб нынче даже 1С)
- Мощное сообщество и развитие
- Удобный пакетный менеджер npm
- Есть тайпчекеры: TypeScript, Flow, Dart…

`ECMAScript и все-все-все`

CUSTIS®

Синтаксис

Сравнительный анализ

ECMAScript и все-все-все

Perl

CUSTIS®

```perl
sub _skip_attrs
{
    my ($tag, $attrs) = @_;
    $tag = lc $tag;
    return "<$tag>" if $tag =~ m!^/!so;
    my ($enclosed) = $attrs =~ m!/$!so ? ' /' : '';
    $attrs = { $attrs =~ /([^\s=]+)=([^\s=\'\"]+|\"[^\"]*\"|\'[^\']*\')/gso };
    my $new = {};
    for (qw(name id class style title))
    {
        $new->{$_} = $attrs->{$_} if $attrs->{$_};
    }
    my %l = (a => 'href', blockquote => 'cite', q => 'cite');
    if ($attrs->{$l{$tag}} && $attrs->{$l{$tag}} !~ /^[\"\']?javascript/iso)
    {
        $new->{$l{$tag}} = $attrs->{$l{$tag}};
    }
    return "<$tag".join("", map { " $_=".$new->{$_} } keys %$new).$enclosed.">";
}
```

- Спецсимволы захватили мир
- Репутация «write-only», развитие умерло

ECMAScript и все-все-все

PHP

CUSTIS®

```php
$isExact = [];
foreach ([ 'line' => 'l', 'cfo' => 'cc' ] as $k => $t)
{
    if (!isset($specified[$k.'_id']) &&
!isset($specified[$k.'_id_exact']) &&
        !isset($groups[$k]) && !isset($groups[$k.'_all']))
        $isExact[] = "$posAlias.${k}_id IS NULL";
    elseif ($lastgrp == $k.'all')
        $isExact[] = "$posAlias.${k}_id=$t.id";
}
foreach ([ 'party', 'account', 'paytype' ] as $k)
    if (!isset($specified[$k.'_id']) && !isset($groups[$k]))
        $isExact[] = "$posAlias.${k}_id IS NULL";
return implode(' AND ', $isExact) ?: '1=1';
```

- Что за $$$$$?
- Репутация г**нокода

ECMAScript и все-все-все

Python

CUSTIS®

```
class FileCache:
  def __init__(self, dir):
    self.dir = dir
    if not os.path.isdir(dir):
      os.mkdir(dir)
  def fn(self, key):
    key = re.sub('([^a-zA-Z0-9_\-]+)', lambda x:
binascii.hexlify(x.group(1)), key)
    return self.dir+'/'+key
  def clean(self):
    t = time.time()
    for fn in os.listdir(self.dir):
      if t > os.stat(self.dir+'/'+fn).st_mtime:
        os.unlink(self.dir+'/'+fn)
```

- Пробелы меняют смысл?!!!!

ECMAScript и все-все-все

Ruby

CUSTIS®

```ruby
module Gitlab
  class SearchResults
    attr_reader :current_user, :query

    def objects(scope, page = nil)
      case scope
      when 'projects'
        projects.page(page).per(per_page)
      when 'issues'
        issues.page(page).per(per_page)
      when 'merge_requests'
        merge_requests.page(page).per(per_page)
      when 'milestones'
        milestones.page(page).per(per_page)
      else
        Kaminari.paginate_array([]).page(page).per(per_page)
      end
    end
```

- projects — переменная? Фигвам. Метод без аргументов.)
- чем он лучше хотя бы питона?

ECMAScript и все-все-все

Go

CUSTIS®

```
func TestChannelStoreSave(t *testing.T) {
        Setup()

        teamId := model.NewId()

        o1 := model.Channel{}
        o1.TeamId = teamId
        o1.DisplayName = "Name"
        o1.Name = "a" + model.NewId() + "b"
        o1.Type = model.CHANNEL_OPEN

        if err := (<-store.Channel().Save(&o1)).Err; err != nil {
                t.Fatal("couldn't save item", err)
        }

        if err := (<-store.Channel().Save(&o1)).Err; err == nil {
                t.Fatal("shouldn't be able to update from save")
        }
```

- Что за смайлики `:= <- & *`? Где мои скобочки?

ECMAScript и все-все-все

Erlang

```
iq_handler(From, _To,
           #iq{type=set, lang = Lang,
               sub_el = #xmlel{name = Operation} = SubEl} = IQ, CC)->
    ?DEBUG("carbons IQ received: ~p", [IQ]),
    {U, S, R} = jid:tolower(From),
    Result = case Operation of
        <<"enable">>->
            ?INFO_MSG("carbons enabled for user ~s@~s/~s", [U,S,R]),
            enable(S,U,R,CC);
        <<"disable">>->
            ?INFO_MSG("carbons disabled for user ~s@~s/~s", [U,S,R]),
            disable(S, U, R)
    end,
    case Result of
        ok ->
            ?DEBUG("carbons IQ result: ok", []),
            IQ#iq{type=result, sub_el=[]};
        {error,_Error} ->
            ?ERROR_MSG("Error enabling / disabling carbons: ~p", [Result]),
            Txt = <<"Database failure">>,
            IQ#iq{type=error,sub_el = [SubEl, ?ERRT_INTERNAL_SERVER_ERROR(Lang, Txt)]}
end;
```

- Ой-ой-ой…
- Специфичен. Классов нет, есть процессы. Но сетевой, да!

ECMAScript и все-все-все

OCaml O_O

```
let log_client_info c sock =
  let buf = Buffer.create 100 in
  let date = BasicSocket.date_of_int (last_time ()) in
  Printf.bprintf buf "%-12s(%d):%d -> %-30s[%-14s %-20s] connected for %5d secs %-10s bw %5d/%-5d %-6s %2d/%-2d reqs "
    (Date.simple date)
  (nb_sockets ())
  (client_num c)
  (
    let s = c.client_name in
    let len = String.length s in
    if len > 30 then String.sub s 0 30 else s)

  (brand_to_string c.client_brand)
  (match c.client_kind with Indirect_address _ | Invalid_address _ -> "LowID"
    | Direct_address (ip,port) -> Printf.sprintf "%s:%d"
          (Ip.to_string ip) port)
  (last_time () - c.client_connect_time)
  (if c.client_rank > 0 then
      Printf.sprintf "rank %d" c.client_rank
      else "")
  (nwritten sock) (nread sock)
  (if c.client_banned then "banned" else "")
  c.client_requests_received
    c.client_requests_sent
  ;
```

- «вроде что-то древнее»
- ПОЛИЗ?!!!!

ECMAScript и все-все-все

Lua

```lua
function Encoder.put(self, chunk)
    if self.buffersize < 2 then
        coroutine.yield(chunk)
    else
        if #self.buffer + #chunk > self.buffersize then
            local written = 0
            local fbuffer = self.buffersize - #self.buffer

            coroutine.yield(self.buffer .. chunk:sub(written + 1, fbuffer))
            written = fbuffer

            while #chunk - written > self.buffersize do
                fbuffer = written + self.buffersize
                coroutine.yield(chunk:sub(written + 1, fbuffer))
                written = fbuffer
            end

            self.buffer = chunk:sub(written + 1)
        else
            self.buffer = self.buffer .. chunk
        end
    end
end
```

- Для полноты картины

ECMAScript и все-все-все

JS

```
TreeGridNode.prototype.setChildren = function(isLeaf, newChildren)
{
    if (!this.tr)
        this.grid.tbody.innerHTML = '';
    else
    {
        var tr = this.tr[this.tr.length-1];
        while (tr.nextSibling && tr.nextSibling._node.level > this.level)
            this.grid.tbody.removeChild(tr.nextSibling);
        if (this.leaf != isLeaf)
        {
            if (isLeaf)
            {
                this.tr[0].cells[0].firstChild.className = 'collapser collapser-inactive';
                removeListener(this.tr[0].cells[0].firstChild, 'click', this._getToggleHandler());
            }
            else
            {
                this.tr[0].cells[0].firstChild.className = this.collapsed ? 'collapser collapser-collapsed' : 'collapser
collapser-expanded';
                addListener(this.tr[0].cells[0].firstChild, 'click', this._getToggleHandler());
            }
        }
    }
    this.leaf = isLeaf;
    this.children = [];
    this.childrenByKey = {};
    this.addChildren(newChildren);
}
```

ECMAScript и все-все-все

CUSTIS®

Синтаксис JS

Имхо вполне нейтральненько.

(неприятных рефлексов вызывать не должен)

CUSTIS®

История JS

- 1995—2004: дремучий лес с партизанами
- 2004—2008: появление AJAX
- 2008+: шустрота и современный период

Дремучий период

CUSTIS®

1995—1997

- Создание и начальная стандартизация
- DOM ещё нет, только «DOM level 0»

  document.forms, document.images

1998

- DOM level 1 (DHTML)
- document.getElementById, все элементы — объекты

2000

- DOM level 2
- События и более-менее современный вид объектов
- pre-AJAX: JSONP, невидимый iframe

Появление AJAX

CUSTIS®

2004

- Firefox 1.0
- XmlHttpRequest
- Первое SPA — Gmail, впервые упомянут термин «AJAX»
- Начало конца дремучего периода JS, как языка для всплывающих баннеров

Совместимость браузеров ещё плохая, так что

- 2006 — jQuery

Современное развитие

CUSTIS®

- 2008 — Google V8
- 2009 — IE8 (M$ очнулся)
- 2009 — node.js (2011 — v0.6)
- 2011 — начало работы над ES6
- 2012 — Angular 1.0
- 2013 — React
- Где-то тут же изоморфность
- 2015 — ES6 принят

Производительность

CUSTIS®

На ЛОРе до сих пор шутят, что «Java тормозит», а что ж тогда JS?

- А ничего — и Java быстрая, и он быстрый. Мамонты видать шутят

  из скриптоты node.js быстрее всех

- Но он же интерпретируемый?

  НЕТ! **Интерпретируемых** языков уже вообще нет.

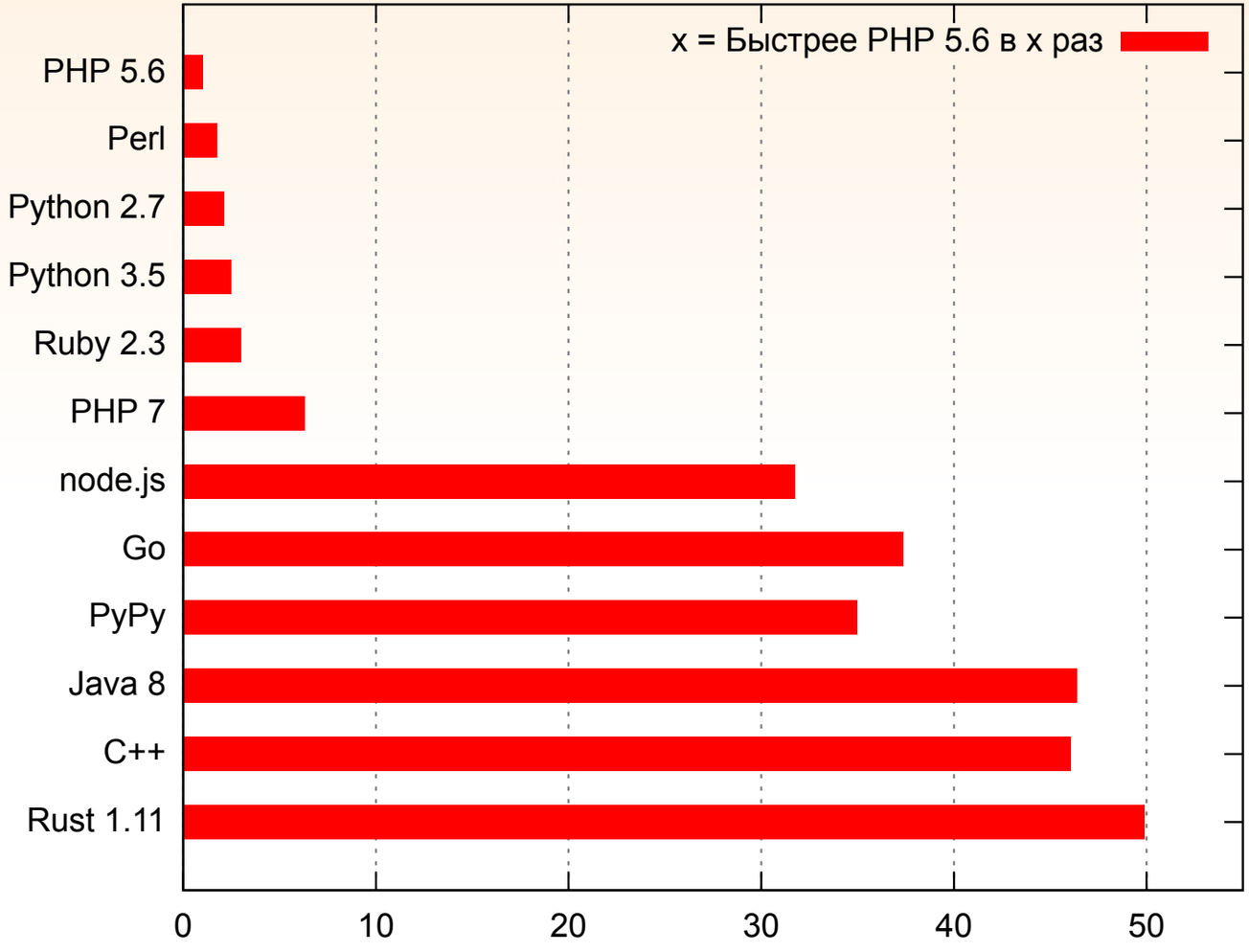
  Ну, разве что bash…

Вычислительный бенчмарк

CUSTIS®

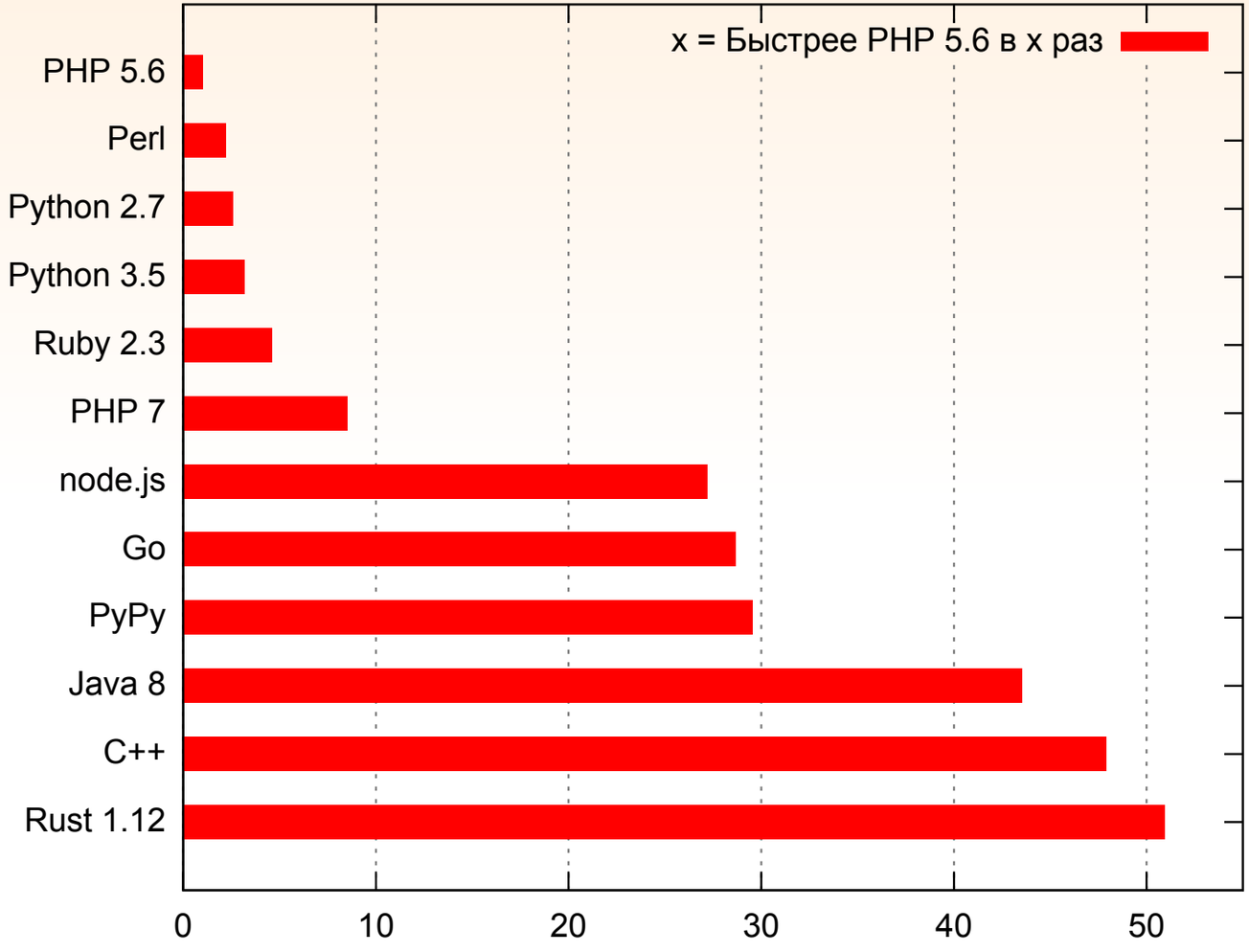
- https://github.com/famzah/langs-performance, время на 10 итераций (i386)
- C++ (g++ 6.1.1) -O2 = 0.92s
- java8 = 0.92s; java6 = 1s *(ручные массивы)*
- PyPy (tracing jit) = 1.25s
- Rust 1.12 = 0.85s! Go 1.7 = 1.14s
- **node.js 4.6 = 1.35s** + баг; nodejs 0.10 = 2.6s
- PHP 7 = 6.7s
- Ruby 2.3 = 14.3s, Python 3.5 = 17s, 2.7 = 20s
- Perl = 24s (у меня 25.6s)
- **PHP 5.6 = 42.5s** :)))))) ахаха, прекрати

Разница между 64 и 32 бит: i386



ECMAScript и все-все-все

Разница между 64 и 32 бит: amd64



ECMAScript и все-все-все

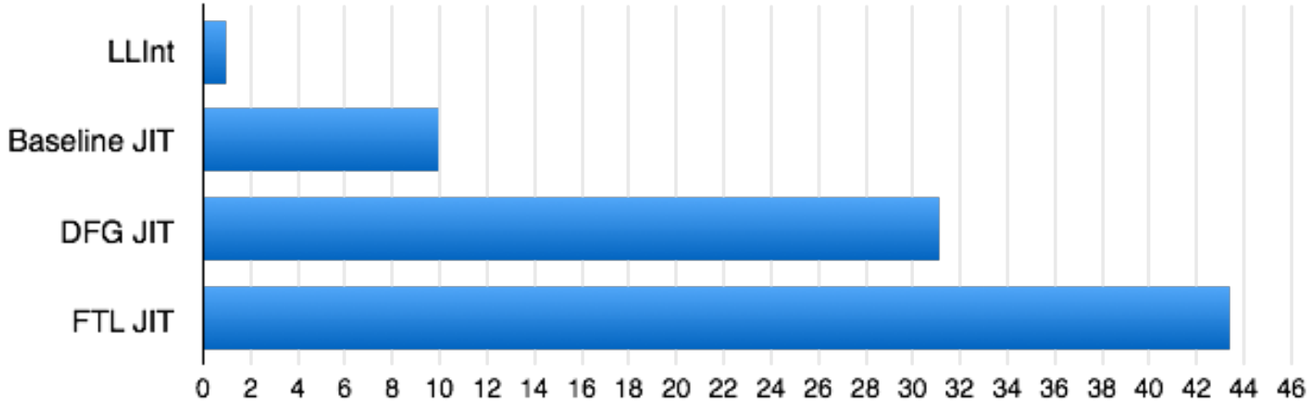
Почему V8 такой быстрый?

CUSTIS®

Потому, что 4-слойный JIT!

- Как уже сказано, интерпретируемых языков нет.
- 1 слой — LLInt, интерпретатор байткода (быстрый старт)
- 2 слой — Baseline JIT
- 3 слой — DFG (Data Flow Graph) JIT

  здесь появляется типизация
- 4 слой — FTL JIT (~~LLVM~~ B3)



[1]

`ECMAScript и все-все-все`

Ключевые слова о том, как это всё устроено

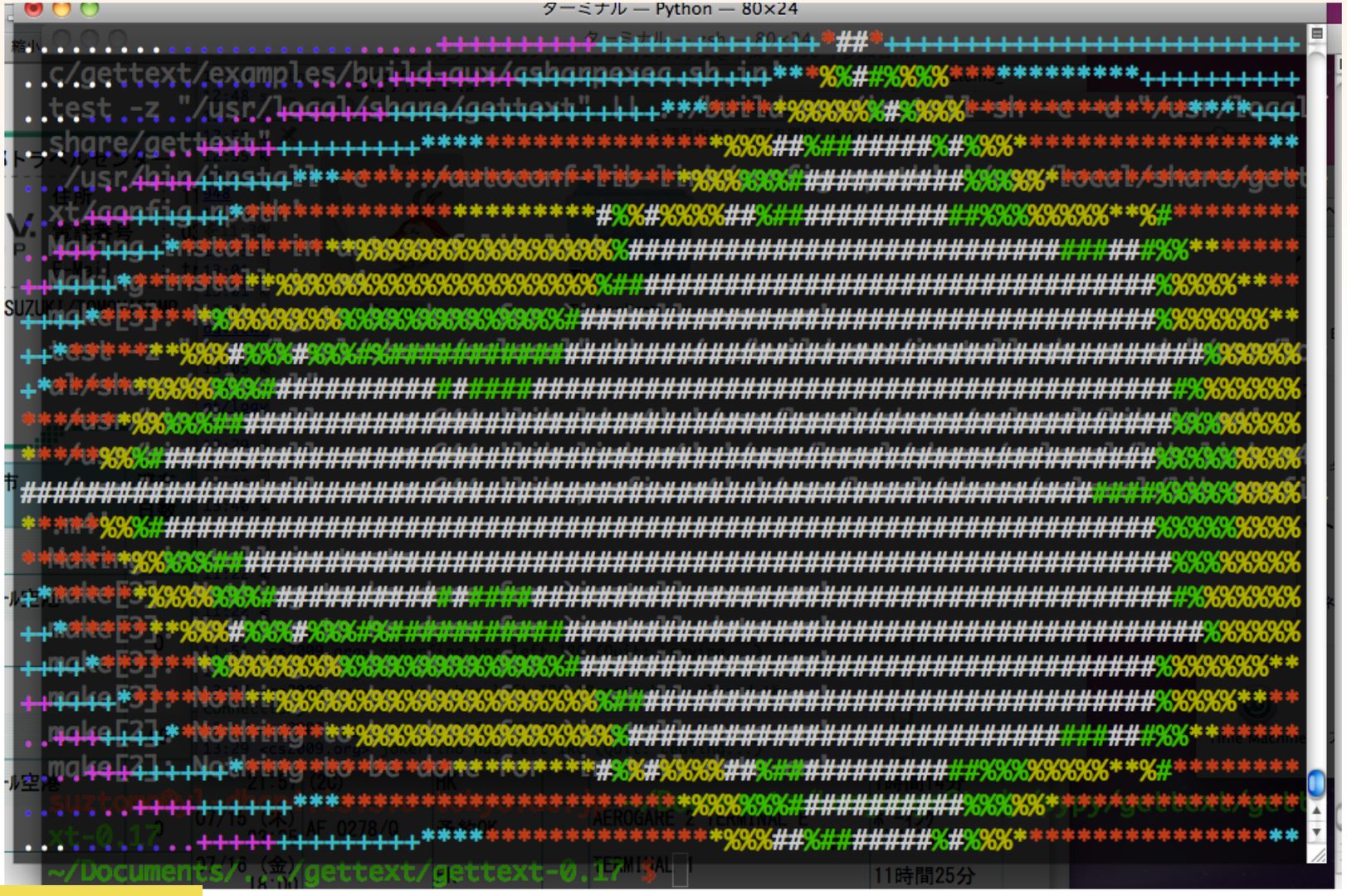
CUSTIS®

- Какой бывает JIT?
  - method-based jit (JVM)
  - tracing jit (PyPy, TraceMonkey)
- Девиртуализация
- Ускорение поиска в хеше (Lua)
- OSR (On-Stack Replace)

Отступление: PyPy

Трассирующий JIT-компилятор для Python, очень медленный

Рисует множество Мандельброта при сборке



ECMAScript и все-все-все

LLVM

CUSTIS®

LLVM (http://llvm.org), ранее «Low Level Virtual Machine»

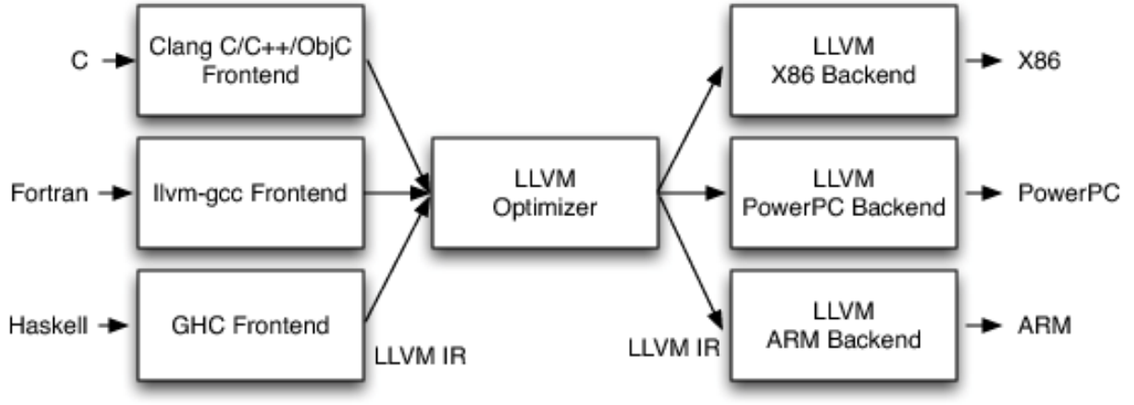
- Набор библиотек для построения компиляторов/интерпретаторов
- Модульный

  исходник → фронтенд (ЯП) → LLVM IR (SSA)

  IR → оптимизатор LLVM → IR

  IR → бэкенд → машинный код

- А также сами компиляторы (в первую очередь C/C++/ObjC: Clang)
- На LLVM сделаны компилятор шейдеров Radeon и OpenCL



ECMAScript и все-все-все

SSA

CUSTIS®

Пример LLVM IR:

```
  store i32 1, i32* %e, align 4
  br label %4
; <label>:4                                       ; preds = %29, %0
  %5 = load i32* %a, align 4
  %6 = load i32* %b, align 4
  %7 = add nsw i32 %5, %6
  store i32 %7, i32* %c, align 4
  %8 = load i32* %c, align 4
  %9 = load i32* %a, align 4
  %10 = sub nsw i32 %8, %9
  store i32 %10, i32* %d, align 4
  %11 = load i32* %d, align 4
  %12 = icmp ne i32 %11, 0
  br i1 %12, label %13, label %14
; <label>:13                                      ; preds = %4
  br label %20
```

ECMAScript и все-все-все

V8 JIT

CUSTIS®

- Первый раз функции запускаются в LLInt
- 6 вызовов либо 100 повторов строки

  → OSR в Baseline JIT
- C*66 вызовов либо C*1000 повторов строки

  → OSR в DFG JIT

  C ~ 1, больше для больших функций
- Нарушение type guard в DFG

  → OSR обратно в Baseline JIT
- C*6666 вызовов либо C*100000 повторов строки

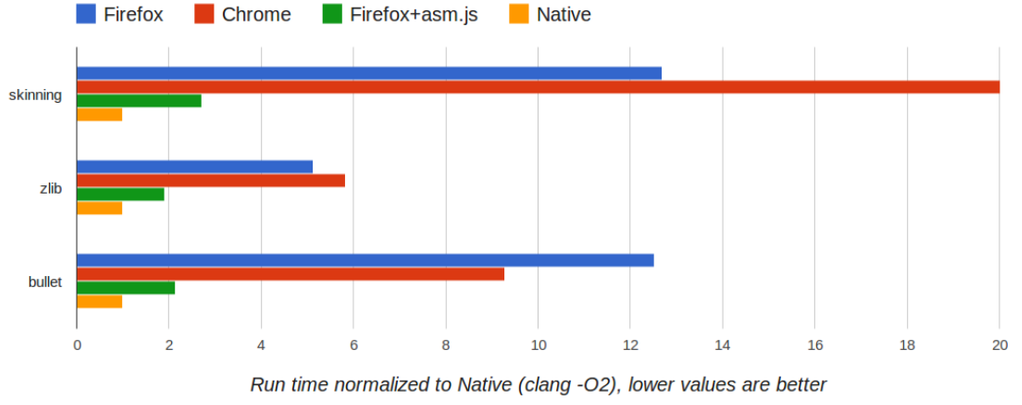
  → OSR в LLVM/B3 JIT

Мало?

CUSTIS®

asm.js (презентация)

```
function strlen(ptr) { // calculate length of C string
  ptr = ptr|0;
  var curr = 0;
  curr = ptr;
  while (MEM8[curr]|0 != 0) {
    curr = (curr + 1)|0;
  }
  return (curr - ptr)|0;
}
```



Run time normalized to Native (clang -O2), lower values are better

ECMAScript и все-все-все

Производительность — итого

CUSTIS®

- Итого, V8 — «смешанный» JIT
- В Firefox — тоже всё шустро
- Чакра Наделлы тоже очень похожа на V8
- Постоянная битва :) https://arewefastyet.com/
- Но язык — ещё не всё! Ещё есть:

  В браузере — DOM (медленный в старых Firefox при формально быстром JS)

  На сервере — ввод/вывод

I/O bound, а не CPU bound

CUSTIS®

- Типичные веб- и бизнес- приложения сетевые

  «получить из одного места (базы, кэша) и переложить в другое / отдать клиенту»

  nb: все наши веб-фреймворки ни фига не делают :)

  nb: иначе PHP-сайтики были бы неюзабельны

- Иногда — **очень сетевые** (C10K / C1M)

  чисто русский термин «хайлоад»

  10gbit ethernet уже среди нас

- ⇒ что же делать?..

CUSTIS®

Событийные машины!

("event loop")

Как вообще обрабатываются соединения клиентов?

CUSTIS®

Все писали сетевой сервер на C? :)

Обычный **(блокирующий)** ввод/вывод:

- forking: socket(), accept(), fork(), дочерний процесс работает с клиентом
- threading: socket(), accept(), создаём поток, дочерний работает с клиентом
- prefork / thread pool: создаём N процессов/потоков заранее
- Потоки и процессы — объекты ОС

    разница только в изоляции памяти

    1000 потоков — уже тяжело (context switch)

    «Проблема C10K» (обработать 10000 соединений на 1 сервере)

`ECMAScript и все-все-все`

Событийная машина

CUSTIS®

Неблокирующий ввод/вывод:

- socket()
- select() / poll() / epoll / kqueue

  говорим ОС: «разбуди, когда в любом из сокетов что-то произойдёт»

  сокет пока один (слушающий)
- новое соединение => добавляем в select и спим дальше
- кто-то прислал запрос => читаем, быстро обрабатываем, отвечаем, спим дальше
- **всё в один поток** (или в несколько по числу ядер CPU)

`ECMAScript и все-все-все`

Так работает Nginx...

CUSTIS®

...и весь современный Web (фронтенды)

(плюс zero-copy и раздача статики через sendfile())

ECMAScript и все-все-все

А если пойти дальше?

CUSTIS®

Кроме HTTP-запросов клиентов ещё есть:

- СУБД/кэши
- Файлы
- REST сервисы
- Websocket’ы
- Сетевые серверы (IMAP?)

Почему бы всё это не обрабатывать в одном цикле?

⇒ Это и будет «событийная машина»

Событийные машины

CUSTIS®

Почти везде опциональная сущность:

- Python: Twisted

  более-менее популярен

- Java: Vert.x, Webbit

  мало кто использует, ынтырпрайз же, а тут хипстота какая-то

  да, внутри wildfly тоже event loop — но только для HTTP

  в JEE — потоки во все поля

- PHP: kak.serpom.po.yaitsam/phpdaemon

  почти никто не использует

- Go: goroutine, но не совсем
- Lua+nginx: ngx_lua/cosocket
- Механизмы разные в разных ОС ⇒ **libevent**, **libev**

`ECMAScript и все-все-все`

Событийная машина node.js

CUSTIS®

- Отличительная черта: **вообще нет блокирующего I/O**

  тут-то колбэки JS и становятся преимуществом

- Реально работает: тест 1M соединений на 1 сервере
- Кроме node.js такое есть только в Erlang

  1M соединений переваривает даже успешнее — на продакшне у WhatsApp

  и gc быстрый (кучи отдельные)

  но… erlang. свои минусы и начинает хотеться монад

- Million RPS Battle

`ECMAScript и все-все-все`

CUSTIS®

Можно писать и на преформе

Но…

«Хорошо быть девочкой в розовом пальто,
можно и не девочкой, но уже не то»

CUSTIS®

Обзор языка

и приколы

ECMAScript и все-все-все

Типы данных

CUSTIS®

Типы данных (что может вернуть typeof):

- **null** (не NULL, всё регистрозависимо)
- **undefined** ("не определено")
- **number**: `1, -100500.05, NaN`
- **boolean**: `true, false`
- **string**: `"hello1", 'hello2'` (всегда Unicode)
- **symbol** — для «скрытых полей» (ES6)
- всё остальное — **object**
  - хеш — объект дефолтного типа: `{ key: "value" }`
  - массив — разновидность объекта (класс Array):
    `[ "value1", 2, 3, "value3" ]`
  - функции — тоже объекты, их typeof = **function**

`ECMAScript и все-все-все`

Переменные

CUSTIS®

```
var a = 123;
```

Новый синтаксис ES6: let/const.

```
let a = 123;
const b = 'hello';
```

- **var** локальны в рамках функции или Global Scope (в браузере — window)
- **let** и **const** локальны, как и положено, в рамках блока

Функции

CUSTIS®

Функции — объекты первого класса в JS.

```
function f(arg)
{
    return arg;
}
var f = function(a)
{
    return a*a;
};
```

Новый синтаксис ES6: arrow functions

```
var f = a => a*a;
var g = (a, b) => a*b;
```

Замыкания

CUSTIS®

Функции в JS являются замыканиями («замыкаются» на текущую область видимости):

```
function a(arg)
{
    var sum = 0;
    var f = function(x) { sum += x; };
    var g = function() { return sum; };
    return [ f, g ];
}
```

Вызывать функцию можно с любым числом аргументов, пропущенные будут undefined, лишние — в arguments.

```
function a()
{
    console.log(arguments[1]); // "abc"
}
a(1, "abc", 3, 4);
```

Прототипы (голый/старый JS)

CUSTIS®

```
function TreeGrid()
{
    // конструктор. тут есть this
}
TreeGrid.prototype.method = function()
{
    // метод класса. тут есть this
    console.log(this);
}

var obj = new TreeGrid();
obj.method();
```

Прототипы

CUSTIS®

(Очень простая альтернатива классам)

- *Объекты «создаются из функций»*
- У каждого объекта есть прототип
- Прототип — тоже объект, в нём функции и «свойства по умолчанию»
- У прототипа может быть свой прототип ⇒ наследование
- object.__proto__ — прототип этого объекта (класс, по сути)
- function.prototype — прототип, создаваемый этой функцией как конструктором

Наследование (голый JS)

CUSTIS®

Для наследования в prototype нужно присвоить объект, __proto__ которого ссылается на базовый класс.

```
function CalendarGrid()
{
    TreeGrid.call(this, arguments); // "super()"
}

CalendarGrid.prototype =
Object.create(TreeGrid.prototype);
CalendarGrid.prototype.constructor = CalendarGrid;
```

this

CUSTIS®

- this передаётся отдельно, как «контекст вызова»
- this можно «подменить» через **function.apply(obj, arguments)** / **function.call(obj, arg1, arg2, …)**
- обычные функции/замыкания **не помнят this** (!)
- arrow functions **помнят** this
- this можно «запомнить» через function.bind() (IE9+ / FF4+)

  также через function.bind() можно сделать **карринг**

  (ну или явно создав замыкание)

this — примеры

CUSTIS®

```
var obj = new TreeGrid();
obj.method(); // this = obj

var f = obj.method;
f(); // this = window

f.apply(obj); // this снова = obj

f = obj.method.bind(obj);
f(); // f привязана к obj (this = obj)

f = obj.method.bind(obj, "hello"); // карринг
f("arg2"); // эквивалентно obj.method("hello", "arg2");

f = obj.method;
f = function() { return f.apply(obj, arguments); }
f(); // то же самое вручную
```

this и замыкания

CUSTIS®

```
TreeGrid.prototype.method = (arg1, arg2) =>
{
    // Так делать нельзя, this всегда = контексту, в котором описан метод (обычно window)
    console.log(this);
};

TreeGrid.prototype.setListener = function()
{
    // Так делать нельзя, this будет = контексту вызова (кликнутому HTML-элементу)
    this.table.addEventListener('click', this.handleClick);
    // И так тоже делать нельзя
    this.table.addEventListener('click', function(event)
    {
        this.handleClick(event);
    });
    // А вот так можно (оп-па!)
    this.table.addEventListener('click', (event) => this.handleClick(event));
    // Ну и вот так тоже можно
    var self = this;
    this.table.addEventListener('click', function(event) { self.handleClick(event); });
};
```

Стандартные конструкции

CUSTIS®

- `if () {} else if () {} else {}` (тело можно однострочное)
- C-подобный for: `for (var i = 0; i < 100; i++) {}`
- `while () {}`
- `do {} while ()`
- break, continue
- C-подобный switch: `switch (x) { case 1: case 2: break; default: alert('x'); }`
- `try { throw 'x'; } catch(e) {} finally {}`

Интересные конструкции

CUSTIS®

- a == b (мягкое сравнение) и a === b (точное сравнение)

  `"" == false, 0 == false, "1" == true, 1 == true, "5" == 5, null == undefined`

- regexp literals: `var re = /<html[^<>]*>/i;`
- break label, continue label
- Цикл по ключам: `for (var i in obj) {}`
- Цикл по значениям (ES6): `for (var i of obj) {}`

```
function a(a1, a2)
{
label:
    for (var i of a1)
        for (var j of a2)
            if (!a2[j])
                continue label;
}
```

Приколы: приведение типов

CUSTIS®

Приколов в JS немного, но они есть.

```
// "+" переопределён для строк
console.log("100"+1 == "1001");
console.log("100"-1 == 99); // можно приводить к числу через a-0
или 1*a

// в ключах хеша true, false, null и undefined превращаются
// в строки "true", "false", "null" и "undefined"
// имхо, в PHP (true="1", false="", null="") сделано логичней
var a = {};
a[false] = 123;
console.log(a["false"] == 123);
```

Приколы: var

CUSTIS®

```
// var локальны в функциях!
function f()
{
    for (var i = 0; i < 100; i++)
    {
        // через 0.5сек 100 раз напечатает "100"
        setTimeout(function() { console.log(i); }, 500);
    }
}

// и определяются как будто в начале!
var a = 42;
function foo() { alert(typeof a); var a = 10; }
foo(); // --> не 42, а undefined!
```

Приколы: неоднозначный разбор

CUSTIS®

```
// эта функция, увы, вернёт undefined ({} станет блоком, а key: меткой)
// хеши начинать строго с той же строки!
function fn()
{
    return
    {
        key: "value"
    };
}

// (function() {})() - определение + вызов функции, типично, чтобы не засорять контекст
// точки с запятой опциональны, но лучше их ставить. иначе:
A.prototype.m = function()
{
} /* вот тут нужна ; */

(function() { B.prototype.m = ... })();

// Эквивалентно A.prototype.m = ( ( function(){} ) ( function() { B.prototype.m = ... } )
)();
```

CUSTIS®

ES6

Он же ES2015. А также ES2016, ES2017

http://es6-features.org/, https://babeljs.io/repl/

Фичи ES6

Уже сказал про:

- let/const
- arrow functions
- for .. of

Вычислимые свойства объектов

CUSTIS®

ES6

```
var b = "key";
var a = { [b+"_value"]: "v2" }; // выражение в []
```

ES5

```
var b = "key2";
var a = { key: "value" }; // ключи - только литералы, можно
без кавычек
a[b] = 'v2'; // динамические ключи только так
```

Destructuring

CUSTIS®

«Деструктивное присваивание»

```
var obj = { key: 'abc', value: [ 1, 2 ] };
var { key: k, value: [ a, b ], other: 3 } = obj; // 3
- значение по умолчанию

// можно в параметрах функции!
[ [ 'a', 1 ], [ 'b', 2 ], [ 'c', 3 ] ]
.reduce(function(obj, [ k, v ]) { obj[k] = v; return
obj; }, {});
```

Оператор распаковки

CUSTIS®

```
function f(...args) {}

f(...iterable);

// Кстати, в ES5 нельзя сделать apply конструктору. А тут:
new Object(...args);

// Распаковка массива
let arr = [ ...iterable, 4, 5, 6 ];

// слияние объектов. у каждого фреймфорка было своё - $.extend, Ext.extend,
Object.assign
let merge_obj = { ...obj1, ...obj2, key: "value" };

// Распаковка в массив
[ a, b, ...other ] = [ 1, 2, 3, 4, 5 ];
```

Упрощённые названия ключей

CUSTIS®

ES6

```
obj = { x, y, a() { return 'abc'; } };
```

ES5

```
obj = { x: x, y: y, a: function a() { return
'abc'; } };
```

Классы (!)

CUSTIS®

«Наконец-то», скажете вы.

```
class Shape
{
    constructor (id, x, y)
    {
        this.id = id;
        this.move(x, y);
    }
    move (x, y)
    {
        this.x = x;
        this.y = y;
    }
}
```

Наследование и property

CUSTIS®

```
class Rectangle extends Shape
{
    constructor (id, x, y, width, height)
    {
        super(id, x, y);
        this._width = width;
        this._height = height;
    }
    static default()
    {
        return new Rectangle("default", 0, 0, 100, 200);
    }
    set width(width) { this._width = width; }
    get width() { return this._width; }
    get area() { return this._width*this._height; }
}
```

Нюанс: IE8+, так как при трансляции в ES5 требуют Object.defineProperty().

Генераторы

CUSTIS®

Функции, из которых можно выйти и вернуться. Пожалуй, самая крутая из всех доработок!

```
function* a()
{
    try
    {
        var v = yield 1;
        if (v < 0) yield 2;
        else yield 3;
    }
    catch (e) { yield "error"; }
    return "final";
}

var b = a(); // объект GeneratorFunction. функция ещё не начала выполняться
b.next(); // вернёт { value: 1, done: false }
b.next(-5); // передаст -5 внутрь генератора. и вернёт { value: 2, done: false }
b.throw(new Error("x")); // передаст исключение внутрь генератора. и вернёт { value:
"error", done: false}
b.next(); // вернёт { value: "final", done: true }
```

Чем генераторы круты?

CUSTIS®

Они дают убрать лестницу колбэков! Пример лестницы (node-postgres):

```
const pg = require('pg');
function makeQueries(callback)
{
  var client = new pg.Client();
  client.connect(function(err) {
    if (err) throw err;
    client.query('SELECT $1::text as name', ['brianc'], function (err, result) {
      if (err) throw err;
      client.end(function (err) {
        if (err) throw err;
        callback(result);
      });
    });
  });
}
makeQueries(function(result) { console.log(result.rows[0]); });
```

ECMAScript и все-все-все

А теперь с генераторами

CUSTIS®

Генератор можно приостановить. Пишем обёртку и получаем coroutine:

```
const gen = require('gen-thread');
const pg = require('pg');

function* makeQueries()
{
  var client = new pg.Client();
  // yield подождёт, пока не выполнится автоматически созданный колбэк gen.ef()
  yield client.connect(gen.ef());
  var result = (yield client.query('SELECT $1::text as name', ['brianc'],
gen.ef()))[0];
  yield client.end(gen.ef());
  return result;
}

gen.run(makeQueries(), function(result) { console.log(result.rows[0]); });
```

ECMAScript и все-все-все

Промисы и async/await

CUSTIS®

То же, но стандартно — делается через промисы и async/await.

```
function sleep(millis)
{
    return new Promise(function(resolve, reject) {
        setTimeout(resolve, millis);
    });
}
async function f()
{
    await sleep(500);
    await sleep(1000);
}
// эквивалентно цепочке промисов:
function f()
{
    return sleep(500).then(result => sleep(1000));
}
```

Это уже не ES6 (2015), а 2016-2017; но Babel всё равно их поддерживает (и транслирует в генераторы).

Поддержка Promise

CUSTIS®

API с колбэками надо оборачивать. Это нетрудно, но надо знать, куда ставить колбэк:

```
function wrap(fn, ...args)
{
    return new Promise(function(resolve, reject)
    {
        try { fn(resolve, ...args); }
        catch (e) { reject(e); }
    });
}
async function test()
{
    await wrap(setTimeout, 500);
}
```

Нюансы с исключениями

CUSTIS®

Нюанс 1: Promise'ы nodejs глотают исключения

Решение — Bluebird, он бросает **Unhandled rejection error**

Нюанс 2: У асинхронных исключений нет вменяемого стека.

Стеки в духе:

```
    at Connection.parseE (node_modules/pg/lib/connection.js:554:11)
    at Connection.parseMessage (node_modules/pg/lib/connection.js:381:17)
    at Socket.<anonymous> (node_modules/pg/lib/connection.js:117:22)
    at emitOne (events.js:77:13)
    at Socket.emit (events.js:169:7)
    at readableAddChunk (_stream_readable.js:146:16)
    at Socket.Readable.push (_stream_readable.js:110:10)
    at TCP.onread (net.js:523:20)
```

Решение — опять-таки Bluebird + `Promise.config({ longStackTraces: true })`.

gen-thread

CUSTIS®

- Ну, или забить пока на промисы
- Юзать генераторы и мой gen-thread (в промисы он тоже умеет)

```
yield gen.p(<PROMISE>);
```

Но в итоге, конечно, все перейдут на Promise.

Модули ES6

CUSTIS®

```
// lib/math.js
export function sum (x, y) { return x + y };
export var pi = 3.141593;
// lib/mathplusplus.js
export * from "lib/math";
export var e = 2.71828182846;
export default (x) => Math.exp(x);
// someApp.js
import exp, { pi, e } from "lib/mathplusplus";
console.log("e^{π} = " + exp(pi));
```

Оговорка: чёткой уверенности, что это лучше CommonJS, у меня нет. Но гибче, да.

Template strings

CUSTIS®

PHP-подобная строковая интерполяция.

```
let name = "John";
`Hello, ${name}`;
```

Кастомная интерполяция (для DSL, видимо):

```
get`http://example.com/foo?bar=${bar + baz}&quux=${quux}`;
// эквивалентно:
get([ "http://example.com/foo?bar=", "&quux=", "" ], bar +
baz, quux);
```

ECMAScript и все-все-все

Proxy

CUSTIS®

```
let target = { foo: "Welcome, foo" };
let proxy = new Proxy(target,
{
    get (receiver, name)
    {
        return name in receiver ? receiver[name] : `Hello,
${name}`;
    }
});
proxy.foo === "Welcome, foo";
proxy.world === "Hello, world";
```

Функции, локальные в блоках

CUSTIS®

(Block-scoped functions)

```
{
    function foo () { return 1 }
    foo() === 1
    {
        function foo () { return 2 }
        foo() === 2
    }
    foo() === 1
}
```

Итераторы для for .. of

CUSTIS®

```
let fibonacci = {
    [Symbol.iterator]() {
        let pre = 0, cur = 1;
        return {
            next() {
                [ pre, cur ] = [ cur, pre + cur ];
                return { done: false, value: cur };
            }
        };
    }
}
for (let n of fibonacci) { if (n > 1000) break; console.log(n); }
```

Обзор инструментов

- Пакетные менеджеры
- Трансляторы
- Сборка, преобразования и минификация
- Отладка, профилировка
- Редакторы, IDE

Системы модулей

CUSTIS®

- AMD (RequireJS): Asynchronous Module Definition
- CommonJS (node.js)
- UMD (AMD+CommonJS, поддерживается и там, и там)
- Модули ES6 (уже рассказал)

Наиболее актуальны CommonJS и ES6.

CommonJS

CUSTIS®

ИМХО — самый краткий и удобный синтаксис.

```
const pg = require('pg');
const gen = require('gen-thread');

module.exports.method = function() { /*...*/ };

// или даже:
module.exports = MyClass;

function MyClass()
{
}
/*...*/
```

AMD

CUSTIS®

```
define(['jquery', 'underscore'], function ($, _) {
    // methods
    function a(){};    // private because it's not returned (see
below)
    function b(){};    // public because it's returned
    function c(){};    // public because it's returned

    // exposed public methods
    return {
        b: b,
        c: c
    }
});
```

ECMAScript и все-все-все

UMD

CUSTIS®

```
(function (root, factory) {
    if (typeof define === 'function' && define.amd)
        define(['jquery', 'underscore'], factory);
    else if (typeof exports === 'object')
        module.exports = factory(require('jquery'), require('underscore'));
    else
        root.returnExports = factory(root.jQuery, root._);
}(this, function ($, _) {
    function a(){};    // private because it's not returned (see below)
    function b(){};    // public because it's returned
    function c(){};    // public because it's returned
    return {
        b: b,
        c: c
    }
}));
```

Пакетный менеджер NPM

CUSTIS®

- Использует систему модулей CommonJS
- Ставит зависимости **рекурсивно** (node_modules/module1 /node_modules/module2/…)
- Можно их упростить: **npm dedup**

Пользоваться им не просто, а очень просто:

- Установить модуль: npm install [-g] [--save] [--save-dev] module
- Создать package.json для проекта: npm init
- Зарегистрироваться/залогиниться: npm login
- Опубликовать свой модуль: npm publish

Самих модулей столько, что аж страшно.

~~Bower~~

CUSTIS®

- Ещё один пакетный менеджер
- Ставится из npm O_o
- Команды примерно те же: install, init… те же semver’ы
- Основные отличия — AMD и плоское дерево зависимостей…

  то есть как будто вы просто сказали npm dedup

  только свалится, если что-то несовместимо по semver

⇒ Нафиг-нафиг.

Transpiler: Babel/Bublé

CUSTIS®

Как писать на ES6, если он не везде поддерживается?

Ответ — Babel!

- REPL: https://babeljs.io/repl/
- Командная строка: npm install babel-cli (babel-node)
- browserify: babelify

Transpiler — транслятор JS в JS. Другие: Bublé, Traceur.

Трансляторы

CUSTIS®

В JS транслируют всё, что можно и нельзя

- List of languages that compile to JS
- Тайпчекеры: TypeScript, Flow, Dart (язык)
- Kotlin, Ceylon, Scala.js, ClojureScript
- Непосредственно Java: TeaVM
- **C++ (O_O)**: Emscripten (LLVM → JS), Mandreel
- Python: Pyjamas, Transcrypt; Ruby: Opal
- OCaml: Ocsigen/Bucklescript
- Haskell: ghcjs, haste
- И много чего ещё

`ECMAScript и все-все-все`

Сборка, упаковка и минификация

CUSTIS®

Актуальное:

- browserify — упаковка npm-модулей (CommonJS) для браузера
- webpack — универсальная система сборки (AMD / CommonJS / ES6 модулей) + CSS + картинки + whatever
- stealjs — оптимизатор js для многостраничных сайтов
- eslint — проверка на ошибки и стиль
- gulp — запускатор скриптов сборки
- uglifyjs — обфускатор

Чуть старее

CUSTIS®

- grunt — запускатор скриптов сборки (часто grunt+bower)
- bower — пакетный менеджер (рассказал выше)
- requirejs — система загрузки и сборки AMD модулей
- jshint — проверка на ошибки и стиль
- yui-compressor, closure compiler — обфускаторы
- rollup — система сборки для ES6 модулей (используется реже)

IDE, редакторы, отладка

CUSTIS®

- Netbeans
  - отладчик встроен
  - (!) умеет живое обновление кода при отладке
- Atom от GitHub
  - сам написан на node.js + webkit (Electron)
  - напоминает Sublime
  - модульный; отладчик — отдельный пакет
- Visual Studio Code
  - мелкий и мягкий форк Atom’а
  - встроен отладчик и поддержка typescript и C#

Отладка из консоли

CUSTIS®

`nodejs debug app.js;` Есть REPL

```
for (var i = 0; i < 5; i++)
{
    debugger; // брейкпоинт
    console.log(i);
}
console.log("done");
```

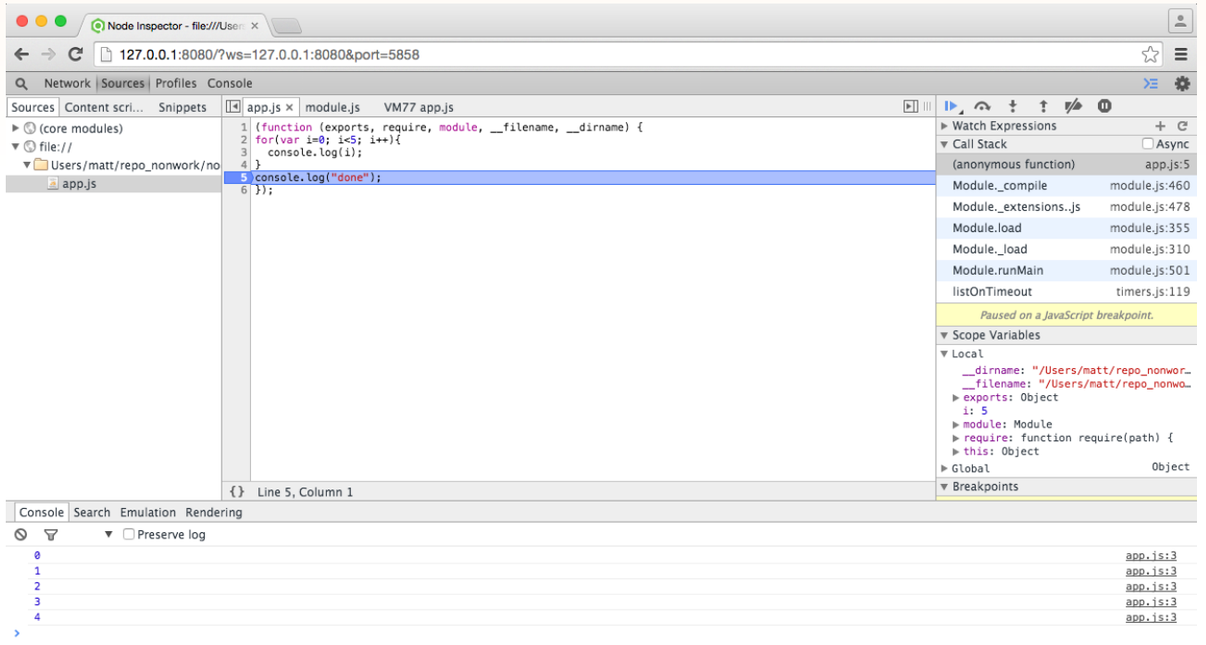
Отладка из node-inspector

CUSTIS®

$ npm install -g node-inspector; node-debug app.js

```
Node Inspector is now available from http://127.0.0.1:8080/?ws=127.0.0.1:8080&port=5858
Debugging `app.js`

Debugger listening on port 5858
```



ECMAScript и все-все-все

Браузерная часть

CUSTIS®

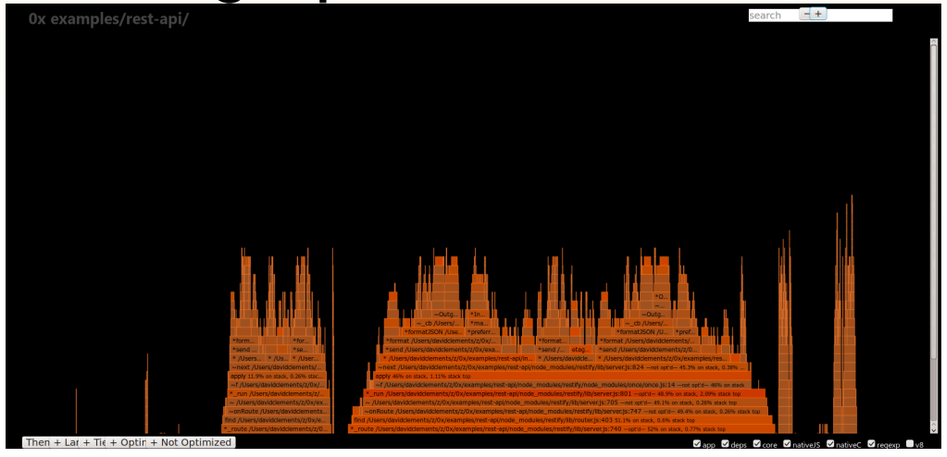
Ну, тут проблем нет совсем

- F12 даже в IE
- watchify
- есть фокусы с live reload (react/redux)
- chrome + netbeans connector

Профилировка

CUSTIS®

- node --prof, node --prof-process
- Профилировка памяти: heapdump
- Flamegraph: 0x анимация



- Ещё заметки

ECMAScript и все-все-все

CUSTIS®

Тестирование

- Для браузера: Karma

  Запускает обычные тесты, но в открытом приложении в браузере

- В целом: Mocha, Chai, Sinon

Тайпчекеры: TypeScript, Flow

CUSTIS®

В целом очень похожи.

- У Flow лучше вывод типов
- У TypeScript есть public/private/protected
- У Flow есть Nullable типы (и он автоматически проверяет на null)
- TypeScript — надмножество JS
- Flow — подмножество JS

TS null

CUSTIS®

```ts
function foo(num: number) {
    if (num > 10) {
        return 'cool';
    }
}
```

```ts
// cool
const result: string = foo(100);
console.log(result.toString());
```

```ts
// still cool?
console.log(foo(1).toString());
// error at runtime: "Cannot read property 'toString' of undefined"
```

ECMAScript и все-все-все

Flow null

CUSTIS®

```
function foo(num: number) {
    if (num > 10)
        return 'cool';
}

// error: call of method `toString`.
// Method cannot be called on possibly null value
console.log(foo(100).toString());

// error: return undefined. This type is incompatible with string
function foo(num: number): string {
    if (num > 10)
        return 'cool';
}
```

TS vs Flow: generics

CUSTIS®

```
class Animal {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
}
class Dog extends Animal {
    // just to make this different from cat
    goodBoyFactor: number;
}
class Cat extends Animal {
    purrFactor: number;
}
```

TS vs Flow: присваивание элементов

CUSTIS®

OK в обоих

```
let cats: Array<Cat> = []; // только коты
let animals: Array<Animal> = []; // только животные

// неа, не кот
cats.push(10);

// неа, не кот
cats.push(new Animal('Fido'));

// круто, кот
cats.push(new Cat('Purry'));

// круто, кот - тоже животное
animals.push(new Cat('Purry'));
```

ECMAScript и все-все-все

TS vs Flow: присваивание коллекций

CUSTIS®

```
// error TS2322: Type 'Animal[]' is not assignable to type 'Cat[]'.
//  Type 'Animal' is not assignable to type 'Cat'.
//    Property 'purrFactor' is missing in type 'Animal'.
//cats = animals;

// во Flow не пройдёт
// она, в TS работает. но теперь animals небезопасны
animals = cats;

// потому что вот это тоже проходит без ошибки
animals.push(new Dog('Brutus'));
animals.push(new Animal('Twinky'));

// ync
cats.forEach(cat => console.log(`Cat: ${cat.name}`));
// Cat: Purry
// Cat: Brutus
// Cat: Twinky
```

CUSTIS®

Обзор фреймворков

- Клиентские фреймворки «старые»
- Клиентские фреймворки «новые»
- Попытки мобильной разработки
- Попытки игровых фреймворков

«Старые»

Заслуживают упоминания:

- jQuery (фиг сдохнет, а надо бы)
- ExtJS (тоже фиг сдохнет)
- R.I.P: Yahoo UI, PrototypeJS
- По-моему, сдыхает: Dojo

jQuery

CUSTIS®

«Меня зовут Мистер Свинья!»

*Если скриптота — рас***дяйство, то jQuery — ВЕРХ рас***дяйства*

Ибо РАСПОЛАГАЕТ к плохому коду:

- Всё в кучу: объект-бог с кучей хелперов $ и такие же модули (например, DataTables)
- `$.extend(), $.ajax(), $("<html>"), $(".class")`
- Селекторы / операции над множествами: `$(".class").each(e => $(e).hide())`

  если элементов не найдётся, no problem, ничего не сдохнет
- Страшные контроллеры, гвоздями прибитые к UI
- Отсутствие возможностей оптимизации при динамической сборке UI

  Пример — WikiEditor

`ECMAScript и все-все-все`

ExtJS

CUSTIS®

- Desktop-like фреймворк, умеет кучу всего
- Фундаментально тормозноват
- Со стилизацией большие трудности

  Ну хоть тема в ExtJS >= 4 стала норм, можно смотреть без рвотных позывов

- По моему опыту — писать на нём НЕ проще и НЕ быстрее, чем на HTML+JS
- Data binding есть, но ограниченный — только свойств (а не иерархии) компонентов

`ECMAScript и все-все-все`

«Новые» (компонентные)

CUSTIS®

- Data binding

  ~~Knockout, Ember, Backbone~~

  упомянем для истории; неактуальны

  AngularJS 1, 2

- Virtual DOM

  React

И те, и другие — по сути, решают задачу шаблонизации на JS.

AngularJS 1

CUSTIS®

- Dirty checking
- Все биндинги 2-way
- Поэтому медленный
- Есть компоненты («директивы») и контроллеры
- На этом уже *можно* писать

Пример

```
<div ng-controller="LoanCalculator">
<div>
   Дата выдачи: <input name="" type="text" ng-model="props.start" /><br />
   Сумма: <input name="" type="text" ng-model="props.total" /><br />
   Процент: <input name="" type="text" ng-model="props.percent" /><br />
   Срок: <input name="" type="text" ng-model="props.months" /> месяцев<br />
   Штраф за просрочку: <input name="" type="text" ng-model="props.fine" /><br />
   Пеня % годовых на просрочку: <input name="" type="text" ng-model="props.penaltyPercent" /><br />
   <input type="button" value="Рассчитать" ng-click="recalc()" />
   <input type="button" value="Сбросить" ng-click="clear()" />
</div>
<table ng-if="payments.length" border="0">
<tr>
   <th>Дата</th>
   <th>Сумма</th>
   <th>Комментарий</th>
</tr>
<tr ng-repeat="payment in payments">
   <td><input name="" type="text" ng-model="payment.date" ng-change="clear_from($index+1)" /></td>
   <td><input name="" type="text" ng-model="payment.total" ng-change="clear_from($index+1)" /></td>
   <td>{{payment.text}}</td>
</tr>
<tr ng-if="clean">
   <th>Всего</th>
   <td>{{sum}}</td>
   <td>{{outsums}}</td>
</tr>
</table>
</div>
```

Angular 2

CUSTIS®

- Пофиксили скорость, разделив 1-way и 2-way биндинги
- Перешли на TypeScript *и всех агитируют*
- Обновились в целом
- Больше «искаропки»
- Но зато тяжелее. Зачем-то тянет за собой RxJS…

React: почему я за React?

CUSTIS®

- JSX: на первый взгляд странно, на самом деле — круто и удобно!
- **Строго однонаправленный** поток данных

  (2-way binding — как чуть сложнее, так проблема)
- Правильная компонентность

  Убирает все сомнения в том, MV-что у нас — MVVM, MVC, M-V-VC, M-V-VC-VM…

  Контроллер занимается тем, чем и должен: работой с данными
- Легковесный, простой, изящный. Учится за 1 вечер. Не принуждает к дополнительным компонентам.

Почему JSX — круто?

CUSTIS®

Даёт писать шаблоны прямо на JS! Удобно и безопасно.

Например, цикл:

- Ember: {{# each}}
- Angular 1: ng-repeat
- Angular 2: ngFor
- Knockout: data-bind="foreach"
- React: ПРОСТО ИСПОЛЬЗУЙТЕ JS :) (обычный for или Array.map())

> Angular 2 continues to put “JS” into HTML. React puts “HTML” into JS.

JSX

```
var MessageInList = React.createClass({
    msgClasses: { unread: 'unread', unseen: 'unseen', answered: 'replied', flagged: 'pinned', sent: 'sent' },
    render: function()
    {
        var msg = this.props.msg;
        return <div data-i={this.props.i} className={'message'+
            (msg.body_text || msg.body_html ? '' : ' unloaded')+
            (msg.flags.map(c => (this.msgClasses[c] ? ' '+this.msgClasses[c] : '')).join(''))+
            (this.props.selected ? ' selected' : '')+
            (msg.thread && this.props.threads ? ' thread0' : '')} onMouseDown={this.props.onClick}>

            <div className="icon" style={{ width: (20+10*(msg.level||0)), backgroundPosition: (10*(msg.level||0))+'px 7px'
}}></div>
            <div className="subject" style={{ paddingLeft: (20+10*(msg.level||0)) }}>{msg.subject}</div>
            {msg.thread && this.props.threads ? <div className={'expand'+(msg.collapsed ? '' : ' collapse')}></div> : null}
            <div className="bullet"></div>
            <div className="from" style={{ left: (21+10*(msg.level||0)) }}>
                {(msg.props.sent ? 'To '+(msg.props.to[0][0]||msg.props.to[0][1]) : (msg.props.from ?
msg.props.from[0]||msg.props.from[1] : ''))}
            </div>
            <div className="size">{Util.formatBytes(msg.size||0)}</div>
            <div className="attach" style={msg.props.attachments && msg.props.attachments.length ? null : { display: 'none'
}}></div>
            <div className="time">{Util.formatDate(msg.time)}</div>

        </div>
    }
});
```

ECMAScript и все-все-все

JSX и Virtual DOM

CUSTIS®

- Как применять изменения быстро?

  (DOM — относительно медленный)

- Храним копию DOM в виде json
- После render() сравниваем
- Изменения применяем к реальному DOM

  ⇒ Быстро и при этом гибко

Компоненты React

CUSTIS®

- Обязательный метод ОДИН: render()
- props и state (XML-атрибуты и внутреннее состояние)
- children (тело элемента)
- getDefaultProps(), getInitialState()
- setState(). нет setProps() и setChildren()! (получает при render родителя)
- componentWillMount(), componentDidMount(), componentWillUnmount()
- componentWillReceiveProps(p), shouldComponentUpdate(p, s)
- componentWillUpdate(p, s), componentDidUpdate(p, s)
- propTypes, mixins, statics

`ECMAScript и все-все-все`

Библиотеки для React

CUSTIS®

- Контроллер: Flux
- Или Redux (2 кб!)
- react-router
- react-router-redux
- HTTP? superagent
- Draft.js (rich editor)

О Redux

CUSTIS®

- «State container»
- Однонаправленный поток данных:

  Store → Компоненты → Действия → Диспетчер → Store
- Компоненты «привязываются» к данным в Store
- И к вызовам действий
- Действие = reducer: (Состояние, Действие) → (НовоеСостояние).

  → Очень чёткое отделение контроллера

Как писать на React

CUSTIS®

Thinking in React

1. Набросать (или взять у дизайнеров) HTML-макет
2. Выделить компоненты

   Помним, что компонент — *единица рендера*
3. Сделать статическую версию на React, всё через props
4. Выделить состояние, определить владельца состояния

   Можно уже брать Flux/Redux

   Presentational vs Controller
5. Научить менять состояние

`ECMAScript и все-все-все`

Прочее

CUSTIS®

- Мобильное
  - React Native: Virtual DOM над нативными компонентами
  - NativeScript: Тоже JS в отдельном потоке и свой XML-язык описания UI
  - Уверен, появятся ещё
- https://html5gameengine.com/
- WebGL: SceneJS, OSG.JS, Blend4Web

CUSTIS®

Демонстрация LikeOpera

Клон Opera Mail на React и node.js

ECMAScript и все-все-все

CUSTIS®

Вопросы

?