

Введение. Динамический язык

Пугачёв Константин (K.V.Pugachev@inp.nsk.su)

2020-09-28

Введение

Современные технологии, методы и языки программирования

- Языки с динамической проверкой типов. ECMAScript
- Системы управления версиями. Git
- Регулярные выражения. PCRE
- Функциональное программирование. Haskell
- Аспектно-ориентированное программирование. AspectJ
- Структурирование данных. XML и его друзья
- Веб-программирование. ECMAScript и его друзья

Пространство и время

- ИЯФ, 508 (КЗ?)
- 2 пары в неделю, 16 недель
- чуть менее, чем половина пар - лекции
 - оглашение информации
 - вопросы по миру идей
- чуть более, чем половина пар - семинары
 - дорешивание и сдача задач
 - вопросы по миру вещей
- <https://www.snd.inp.nsk.su/~pugachev/nsu-prog/mtmpl/>

- Не понял – задай вопрос
- На пары стоит ходить, задачи стоит сдавать
- Сдано всё и вовремя, нет пропусков - 5
- Сдано не всё/невовремя/не было какое-то время - 2, 3, 4

Языки

Парадигм крайне много

- Структурное п.: программа - иерархия блоков
- Императивное п.: программа - набор действий
- Процедурное п.: программа - иерархия процедур
- ОО п.: программа - граф объектов и их взаимодействие
- Функциональное п.: программа - граф функций
- Декларативное п.: программа - описание результата
- Логическое п.: программа - набор логических правил
- ещё парадигмы...

Откуда разнообразие языков

- Язык создавался под некоторую область и имеет средства для работы с ней
 - С и мир UNIX
 - ECMAScript и мир WWW
- Язык создавался с напором на выражение мыслей в определённой форме
 - C++ - через классы и объекты
 - Haskell - через функции и математику
- Язык создавался потому, что предыдущий надоел
 - C++ - потому, что надоел С
 - TypeScript - потому, что надоел ECMAScript

Система/проверка типов

- Строгая/сильная типизация: $1 + 1.0$ – ошибка
- Нестрогая/слабая типизация: $1 + [1] = 2$
- Явная типизация: `var x: int := 1`
- Неявная типизация: `var x := 1`
- Статическая типизация: $1 + [1]$ – ошибка компиляции
- Динамическая типизация: $1 + [1]$ – ошибка исполнения
- Утиная типизация:
`f(x) = x+1 – forall x: defined plus(x, int)`
- Типы как вложенные множества: `class A derives B`
- Автоматический вывод типов:
`int x := f(2); f(x) = x+1 – f: int -> int`

ECMAScript

JavaScript

- 1995, Brendan Eich
- стандарт: ECMA-262, язык ECMAScript
- мультипарадигменный (имп. п. (проц. п., ООП, прот. п.), ФП)
- динамическая слабая неявная типизация без вывода типов
- автоматическое управление памятью
- однопоточная работа с упором на событийно-ориентированную архитектуру и асинхронность
- как бы интерпретируемый (уже давно нет, в V8 джава компилятора)

Пример набора инструментов для работы с JS

- Notepad++ для написания кода
- Chrome, DevTools (F12 button) для исполнения с графикой
- Node.js для исполнения в консольке
- repl.it для игр в вебе

Сферический в вакууме Hello, world в ECMAScript

```
throw new Error('Hello, world!');
```

В ECMAScript нет стандартного универсального способа вывода.

Жизненный Hello, world в ECMAScript

```
console.log('Hello, world!');           // web, Node.js  
alert('Hello, world!');                 // web  
document.write('Hello, world!');        // DOM (web)  
WScript.Echo('Hello, world');           // WSH  
  
import System;  
Console.WriteLine('Hello, world!');     // JScript.Net
```

Основные типы

- **Number** - недлинное целое (32бит) + с плавающей точкой (64бит)
 - 1, 1.0, 1.2e-3, доступны +, -, &, ... - как в C, **, Math.sin(x), Math.exp(x), ...
 - `typeof 1 == 'number', Number('1') == 1`
 - `2 / 3 == 0.66666666666666, 7 / 2 | 0 == 3`
- **String** - строка (ведёт себя как иммутабельная; Юникод по умолчанию)
 - 'abc', "aaa\r\n", "іжак", доступны +, s[1], s.slice(2, 3) и т.д.
 - `typeof 'іжак' == 'string'`
- **Boolean** - логическое значение
 - true, false, доступны &&, ||, !, ... - как в C
 - `typeof true == 'boolean', Boolean('false') == false`

Основные типы

- Undefined - вселенская пустота
 - undefined, хранится во всех неинициализированных значениях
 - `typeof undefined == 'undefined', void 10 == undefined`
- Object - гетерогенный ассоциативный массив
 - `{a: 3, b: 3}`, доступны `o['a']`, эквивалентное `o.a`, `'a' in o` и т.д.
 - `null` - как `undefined`
 - `typeof {} == 'object'`
- Array - гетерогенный массив (Object, в ключах которого строки из цифр)
 - `Array(10), [1, "2", true]`, доступны `a[0]`, `a.length`, `a.slice(2, 3)` и т.д.
 - `typeof [] == 'object'`

Основные типы

- `Function` - функция
 - `function f(x) { return x+1; }, function(x){ return x+1; }, x => x+1`
 - `typeof (x => x+1) == 'function', Function('x', 'y', 'return x+y')`
- `Symbol` - невидимый ключ в ассоциативном массиве
 - `Symbol('a')`, доступны `obj[Symbol('a')]`
 - `typeof Symbol('a') == 'symbol'`
- `BigInt` - длинное целое
 - `BigInt('12342359804250')`, `1234354902n`, доступны `+`, `-`, `**` ...
 - `typeof 123n == 'bigint'`

Значения, функции

```
var    x = 10; // переменная, видна внутри функции  
const y = 20; // константа, видна внутри функции  
let    z = 30; // переменная, видна внутри скобок
```

```
function sum (x, y) {  
    console.log('ha-ha, classic', x, y);  
    return x + y;  
}
```

```
x = 11; // присваивание  
x += sum(y/z, z); // синтаксис в стиле C/Java
```

Автоматические касты

```
'1' + '2'      // '12' (+ кастит к строке, когда не число)
'1' + 2        // '12'
[] + []        // ''
+ '1'          // 1 (унарный плюс кастит к числу)
1 + 2          // 3
2 - '1'        // 1 (арифметические кастят к числу)
2.2 | 0        // 2 (битовые кастят к целому числу)
2.2 , 1        // 1
2.2 || 1       // 2.2 (логические возвращают значение)
2.2 == '2.2'   // true (сравнение кастит к...)
2.2 === '2.2'  // false (===, !== сравнивают без каста)
[] / 2         // 0
!0             // true
null == undefined // true
null === undefined // false
```

Мораль: читать стандарт или не делать два операнда разного типа без надобности.

Что есть истина? Что есть ложь?

- Истина - `true`
- Ложь - `false`
- Приводится к истине `1`, `[1]`, `[]`, `{}`, `"aaa"`, `x => 1`
- Приводится ко лжи `0`, `""`, `null`, `undefined`
- Приведение осуществляют `if`, `!`, функция `Boolean`
- Приведение осуществляют в глубине души `&&`, `||`

Передача по...

- Number, Boolean - значению
- String - значению (фактически - по константной ссылке)
- Array, Object, Function - ссылке

```
var refX = {X: null};  
  
function setX(rx, vx) {  
    rx.X = vx;  
}  
  
setX(refX, 1);  
refX.X // 1
```

Ветвления, циклы

```
if (a || c > 100500) { aaa(); bbb(); }  
else eee(); // можно опускать скобки как в C
```

```
while (1) { console.log('Again'); }
```

```
for (var x=1, i=2; x<i; i++) { ... }
```

```
for (var i in {a: 3, b: 5}) {  
    console.log(i); // 'a' or 'b'  
}
```

```
for (var x of [1, 2, 'too much']) {  
    console.log(x); // 1, 2 or 'too much'  
}
```

Скоупы

```
function () { // {} выпускает var  
  // x == undefined  
  if (0) { var x = 2; }  
  // x == undefined  
  if (1) { var x = 3; }  
  // x == 3  
}
```

```
function () { // функция не выпускает var  
  // unknown x  
  function() { var x = 2; /* x == 2 */ }  
  // unknown x  
}
```

```
function () { // {} не выпускает let  
  // unknown x  
  if (1) { let x = 2; /* x == 2 */ }  
  // unknown x  
}
```

Функция как объект первого класса

```
var fac = function (n) { return n > 0 ? n * fac(n-1) : 1; }  
let sqr = x => x*x;  
  
function applyTwice (func) { // ф. на входе, ф. на выходе  
  // func - в замыкании  
  return x => func(func(x));  
}  
  
console.log(applyTwice(sqr)(2)); // 16
```


Неявный аргумент this

```
function func () {  
    console.log(this);  
}
```

`func ();` *// this - гл. объект или null в strict mode*

```
var obj = {aaa: func};  
obj.aaa(); // this - obj  
obj['aaa'](); // this - obj
```

```
var f = obj.aaa;  
f(); // CAVEAT, PYTHONER!  
      // this - гл. объект или null в strict mode
```

```
var g = obj.aaa.bind(obj);  
g(); // this - obj (как в питоне!)
```

Неявный аргумент this и стрелочная функция

```
let func = () => { console.log(this); };
```

```
func (); // this - null
```

```
var obj = {aaa: func};  
obj.aaa(); // this - null  
obj['aaa'](); // this - null
```

```
var f = obj.aaa;  
f(); // this - null
```

```
var g = obj.aaa.bind(obj);  
g(); // this - null
```

Аргументы по умолчанию, *-паковка массивов и объектов

```
function f(x = 10, y = 20) {  
  return x + y;  
}  
  
console.log(f()); // 30  
console.log(1, 1); // 2  
console.log(...[1, 2]); // 3  
  
function g({x = 10, y = 20} = {}) {  
  return x + y;  
}  
  
g({x: 1, y: 2}); // 3  
g(); // 30
```

Объекты и прототипы

```
function cube (a) {  
  this.size = a; // принадлежит объекту  
  this.getSize = () => a; // принадлежит объекту  
}  
  
cube.prototype.dim = 3; // принадлежит всем кубам  
cube.anotherDim = 10; // принадлежит конструктору кубов  
  
cube.prototype.vol = function() { // принадлежит всем кубам  
  return Math.pow(this.size, this.dim);  
};  
  
let c = new cube(2);  
c.size // 2  
c.getSize() // 2  
c.dim // 8  
c.vol() // 3
```

Объекты и прототипы

```
// в `с` лежат `size` и `getSize`  
'size' in с // true  
с.hasOwnProperty('size') // true  
с.size = 10 // изменили размер куба  
  
// в `cube.prototype` лежат `dim` и `vol`  
// если свойства нет в с, оно берётся из cube.prototype  
'dim' in с // true  
с.hasOwnProperty('dim') // false  
  
с.dim = 4 // добавили dim в с  
с.dim // 4  
с.hasOwnProperty('dim') // true  
  
cube.prototype.dim // 3
```

Цепочка прототипов

```
function A() {}  
A.prototype.x = 1;  
  
function B() {}  
B.prototype = new A;  
  
function C() {}  
C.prototype = new B;  
  
let o = new C;  
o.hasOwnProperty('x')    // false  
o.constructor            // функция A  
  
o instanceof A           // true  
o instanceof B           // true  
o instanceof C           // true
```

Ассоциативный массив или объект?

И то, и другое, причём можно выстрелить себе в ногу.

```
var o = {a: 2};

'a' in o // true
o.hasOwnProperty('a') // true

Object.prototype.b = 3; // сломали in
'b' in o // true
o.hasOwnProperty('b') // false

'hasOwnProperty' in o // true
o.hasOwnProperty('hasOwnProperty') // false

o['hasOwnProperty'] = 4; // сломали hasOwnProperty
o.hasOwnProperty('a') // ошибка: 4 – не функция
```

Ассоциативный массив

```
var o = Object.create(null); // объект без прототипа
o.a = 2;
Object.prototype.b = 3;

'a' in o // true
'b' in o // false
'hasOwnProperty' in o // false
```

В таком ассоциативном массиве спокойно проверяем наличие ключа через `in`.

ООП - Обёртка над прототипами

```
class cube {  
  constructor(a) {  
    this.size = a;  
    this.getSize = () => a;  
  }  
  vol () { return Math.pow(this.size, cube.dim); }  
  set volume(v) { this.size = Math.pow(v, 1/3); }  
  static dim = 3;  
}
```

```
let c = new cube (2);  
c.size           // 2  
c.vol()          // 8  
c.volume = 64  
c.size           // 4  
c.constructor    // cube  
c.constructor.dim // 3  
c.constructor.prototype.vol.call(c)
```

ООП - Обёртка над прототипами

```
class redcube extends cube {  
  constructor (a) {  
    super(a);  
    this.color = 'red';  
  }  
  getColor () { return this.color }  
}  
  
let rc = new redcube (2);  
rc.hasOwnProperty('size') // true  
rc.hasOwnProperty('color') // true  
rc.constructor.prototype.hasOwnProperty('getColor') // true  
rc.constructor.prototype.hasOwnProperty('vol') // false  
'vol' in rc.constructor.prototype // true
```

Точки с запятой

```
var a = 3; // хочу – ставлю
var b = 4  // хочу – не ставлю

var c = 3 + // парсер умный,
  4         // он поймёт

return // упс... будет return undefined
  5    // а это оператор 5, никогда не выполнится
```

Много что - объект

```
2..toString() // можно звать методы
```

```
new Number(2) // можно боксить
```

```
new Number(2) instanceof Number // true
```

```
typeof new Number(2) // 'object'
```

```
Number(2) // можно кастовать (боксить и сразу анбоксить)
```

```
Number(2) instanceof Number // true
```

```
typeof Number(2) // 'number'
```

```
2..constructor.prototype.twice = function() {
```

```
    return this * 2;
```

```
};
```

```
2..twice() // 4
```

```
Boolean.prototype.is = {'bool': 111111};
```

```
true.is.bool // 111111
```

Reflection и т.п.

```
var a = {'a': 1};
```

```
Object.defineProperty(a, 'b', {  
  enumerable: false,  
  configurable: false,  
  writable: false,  
  value: 2  
});
```

```
for (var i in a) i; // 'a' и всё; 'b' не enumerable  
'b' in a // true
```

```
Object.keys(a) // ["a"]
```

```
Object.getOwnPropertyNames(a) // ["a", "b"]
```

Методы Array.prototype

- slice: `[1,2,3,4,5].slice(2,3) // [3]`
- push: `var a=[1]; a.push(2); a // [1,2]`
- pop: `var a=[1,2]; a.pop() // 2`
- map
 - `[1,2,3].map(x => x+1) // [2,3,4]`
 - `[1,2,3].map(x => 1) // [1,1,1]`
 - `[,, undefined].map(x => 1) // [, , 1]`
- reduce
 - `[1,2,3].reduce((x, y) => x+y, 0) // 6`
 - `[1,2,3].reduce((x, y) => x+y, '!') // "!123"`
- filter: `[1,2,3].filter(x => x % 2) // [1,3]`
- не метод не прототипа length
 - `[1,2,3].length // 3`

Жонглирование коллекциями - полезное умение

```
cat text # весь текст  
cat text | grep lololo # только то, где есть "lololo"  
cat text | grep lololo | grep azaza # ... "lololo" и "azaza"
```

```
SELECT filtered.line FROM (  
    SELECT line FROM text WHERE line like '%lololo%'  
) as filtered  
WHERE filtered.line like '%azaza%';
```

```
const fs = require('fs');  
const arr = String(fs.readFileSync('text')).split('\n');  
console.log(arr  
    .filter(x => x.indexOf('lololo') >= 0)  
    .filter(x => x.indexOf('azaza') >= 0));
```

reduce - универсальный цикл

```
const map = (f, xs) =>  
  xs.reduce((xs, x) =>  
    xs.concat([f(x)]), []);
```

```
const reverse = xs =>  
  xs.reduce((xs, x) =>  
    [x].concat(xs));
```


Информация

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://developer.mozilla.org/ru/docs/Web/JavaScript>
- <https://learn.javascript.ru/>