Парсеры - это спарта

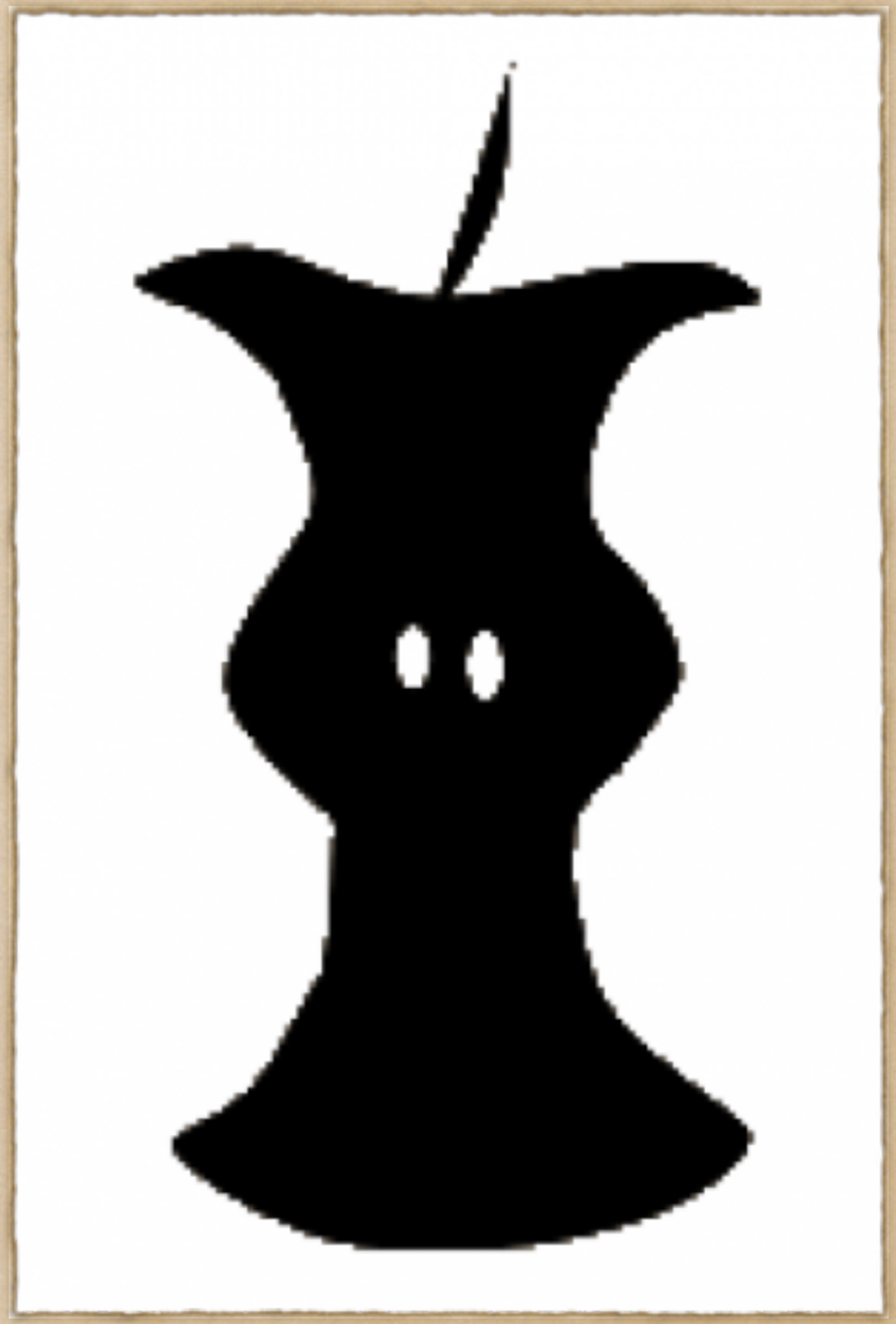
Алексей Охрименко
IPONWEB

http://www.devconf.ru


CAUTION
THIS IS SPARTA

Алексей Охрименко

Twitter: @Ai_boy



{ }
BACKEND
DEVELOPER

</>
FRONTEND
DEVELOPER

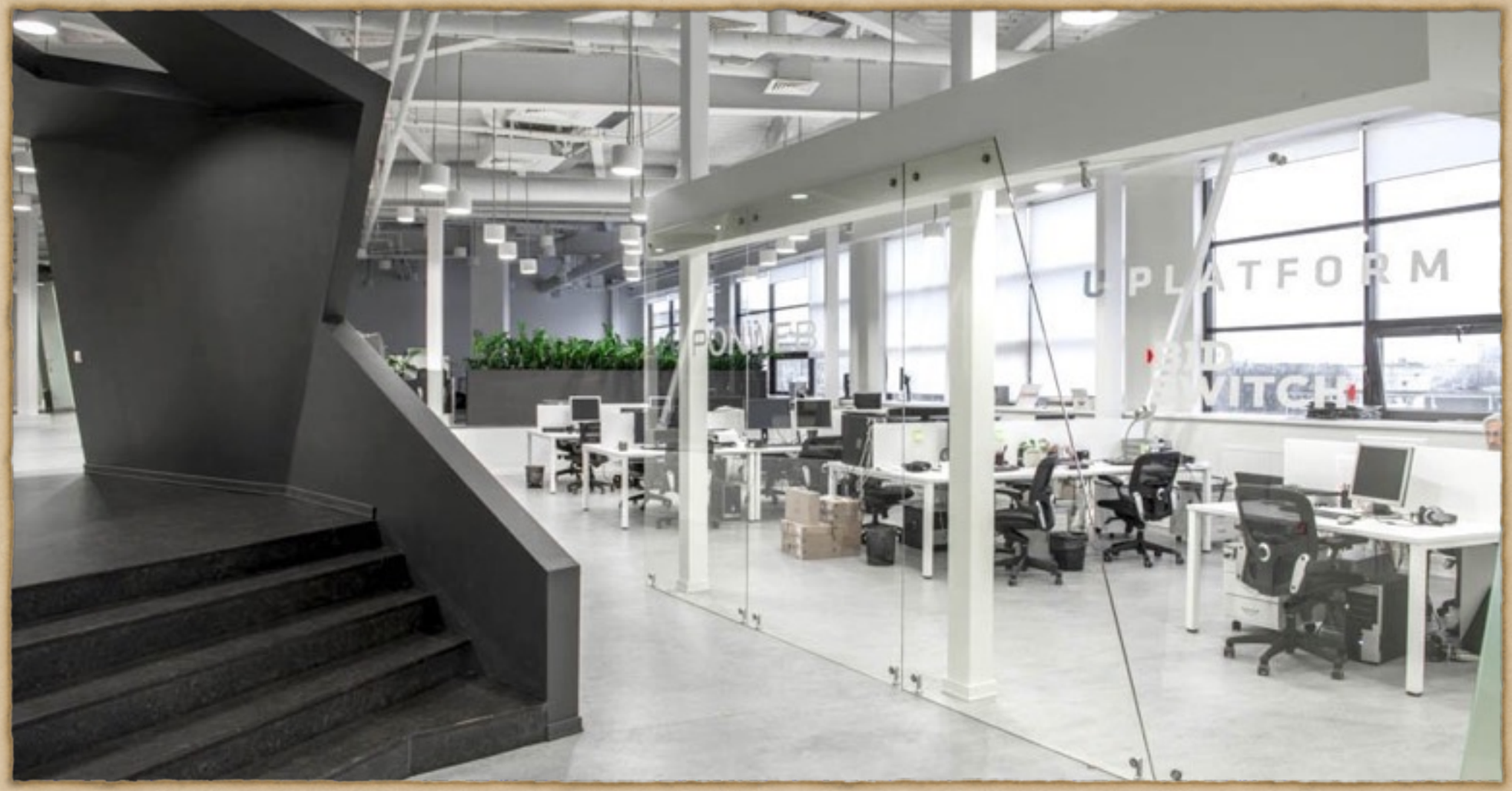
ПРОСТИ, ЧЕБУРАШКА

ты родился в Спарте

Lionidus & HIGHLOAD 2015

Тимур

infosystems jet



IPONWEB

RTB


IPONWEB
IPONWEB
IPONWEB
ВСЕ
ОЧЕНЬ
СЛОЖНО

Вы готовы?

```
<input type="email">
<input type="email" multiple>
```

Live Demo

" "@[IPv6:2001:db8::1]

user@gmail.com

Отправить

Firefox	Safari	Safari Mobile	Chrome	Opera	IE
4+	5+	iOS 3.1+	10+	10.6+	10+
✔	~	~	✔	✔	✔

value = e-mail address

A single e-mail address.

Value: Any string that matches the following [ABNF] production:

```
1*( atext / "." ) "@" ldh-str 1*( "." ldh-str )
```

…where *atext* is as defined in [RFC 5322], and *ldh-str* is as defined in [RFC 1034].

That is, any string which matches the following regular expression:

```
/^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/
```

Examples:

```
foo-bar.baz@example.com
```

RFC 822, 2822, 5322

- https://tools.ietf.org/html/rfc822 (1982 год)
- https://tools.ietf.org/html/rfc2822 (2001 год)
- https://tools.ietf.org/html/rfc5322 (2008 год)

http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html

Regular expression Denial of Service - ReDoS

*This is an **Attack**. To view all attacks, please see the Attack Category page.*

Last revision (mm/dd/yy): **11/9/2015**

Introduction

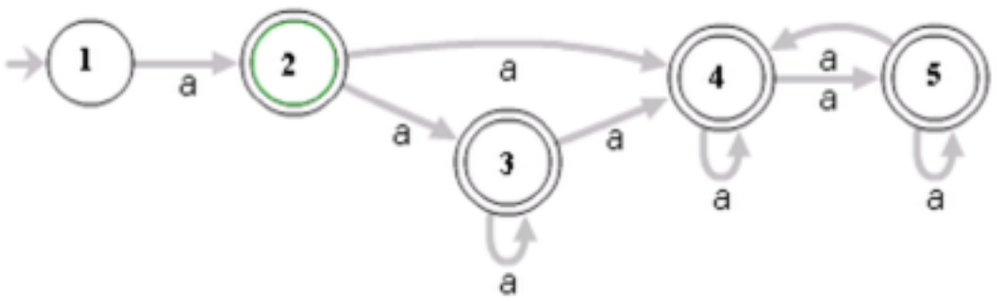
The **Regular expression Denial of Service (ReDoS)** is a Denial of Service attack, that exploits the fact that most Regular Expression i
work very slowly (exponentially related to input size). An attacker can then cause a program using a Regular Expression to enter these

Description

The problematic Regex naïve algorithm

The Regular Expression naïve algorithm builds a Nondeterministic Finite Automaton (NFA), which is a finite state machine where for e
states. Then the engine starts to make transition until the end of the input. Since there may be several possible next states, a determinis
paths (if needed) until a match is found (or all the paths are tried and fail).

For example, the Regex ***^(a+)+$*** is represented by the following NFA:



For the input ***aaaaX*** there are 16 possible paths in the above graph. But for ***aaaaaaaaaaaaaaaaX*** there are 65536 possible paths, and t
where the naïve algorithm is problematic, because it must pass on many many paths, and then fail.

Notice, that not all algorithms are naïve, and actually Regex algorithms can be written in an efficient way. Unfortunately, most Regex eng
Regexes with "special additions", such as back-references that cannot be always be solved efficiently (see **Patterns for non-regular la**

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal
Presentations
Press
Projects
Video
Volunteer

Reference
Activities
Attacks
Code Snippets
Controls
Glossary
How To...
Java Project
.NET Project

```
(function () {

    it("a", function() {
        console.log(1);
    });

    it("z", function() {
        console.log(1);
    });

}());
```

```
(function () {

    it("a", function() {
        console.log(1);
    });

    it("z", function() {
        console.log(1);
    });

}());
```

```
(function () {

    xit("a", function(){})

    it("z", function() {
        console.log(2);
    });

}());
```

```
(function () {

    it("a", function() {
        console.log(1);
    });

    it("z", function() {
        console.log(1);
    });

}());
```

```
(function () {

    xit("a", function(){})

    it("z", function() {
        console.log(2);
    });

}());
```

```
(function () {
    xit('a - OVERRITEN', function () {
    });
    it('z - OVERRITEN', function () {
        console.log(2);
    });
}());
```

https://github.com/rdio/jsfmt

Example

```
var jsfmt = require('jsfmt');
var fs = require('fs');

var js = fs.readFileSync('each.js');

js = jsfmt.rewrite(js, "_.each(a, b) -> a.forEach(b)");
```

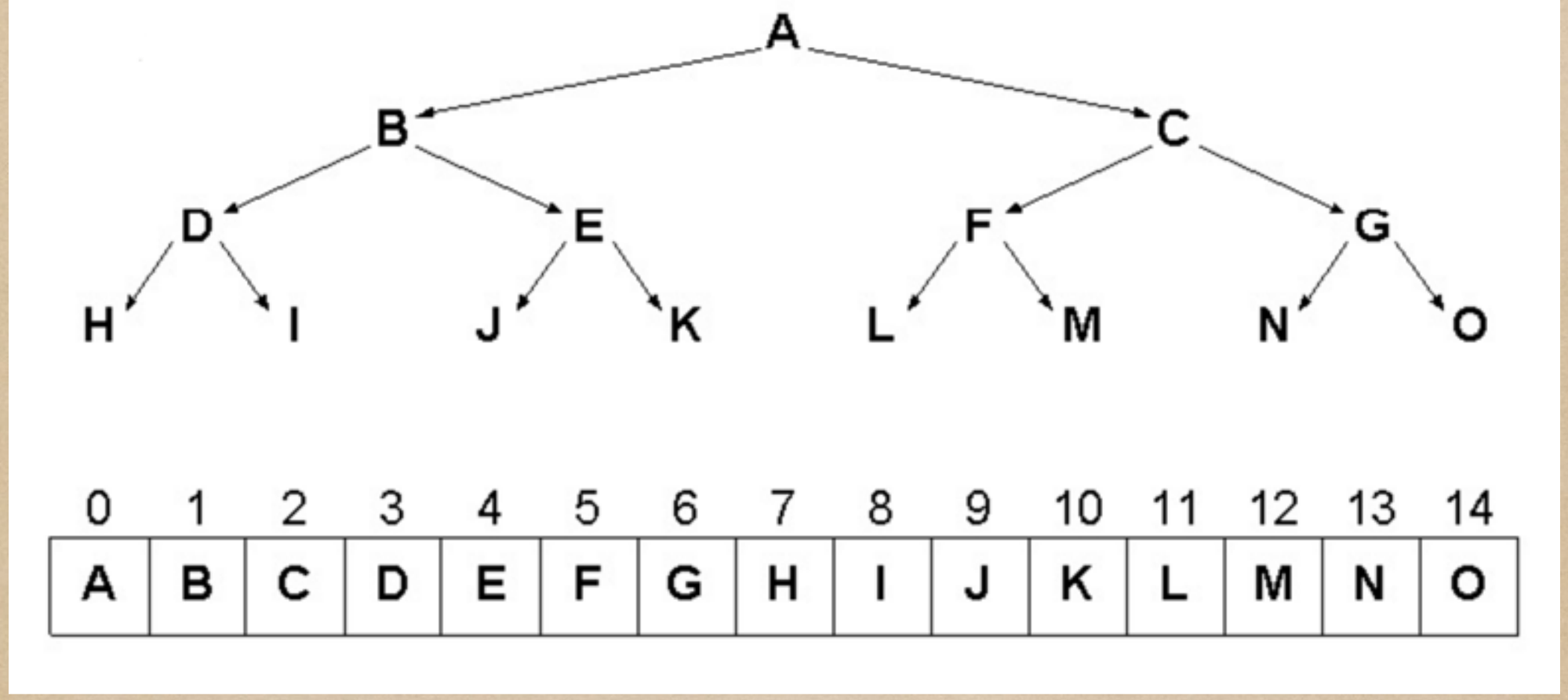
EasyHype

AutoNGConvertor

Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar

В основе любой сложной задачи стоит неправильно заданный вопрос.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
A B C D E F G H I J K L M N O

Что нам нужно?

- Парсер
- Грамматика
- Абстрактная формальная грамматика
- Система описания синтаксиса

Парсер

```
var input = 'a';
var position = 0;

function string(rule) {
    if (input[position] === rule) {
        return {
            type: 'string',
            match: rule,
            start_position: position,
            end_position: position + rule.length
        }
    } else {
        return false;
    }
}
```

```
var ast = string('a');

{
  "type": "string",
  "match": "a",
  "start_position": 0,
  "end_position": 1
}
```

Каррирование

```
function sum(a, b) {
    return a + b;
}

sum(1,2) // 3
```

```
function sum(a) {
    return function(b) {
        return a + b;
    }
}

sum(1)      // function
sum(1)(2)   // 3
```

Рекурсивное Каррирование

```
function A() {
    return function () {
        return B();
    }
}

function B() {
    return function () {
        return A();
    }
}

A();  // Infinite loop
```

```
function rec(callback) {
    return function() {
        return callback().apply(this, arguments)
    }
}
```

```
function A() {
    return function () {
        return rec(B);
    }
}

function B() {
    return function () {
        return rec(A);
    }
}

A();  // Works!
```

```
function string(rule) {
    return function() {
        if (input[position] === rule) {
            return {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position += rule.length
            }
        } else {
            return false;
        }
    }
}
```

```
function string(rule) {
    return function() {
        if (input[position] === rule) {
            return {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position += rule.length
            }
        } else {
            return false;
        }
    }
}
```

```
function string(rule) {
    return function() {
        if (input[position] === rule) {
            return {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position += rule.length
            }
        } else {
            return false;
        }
    }
}
```

```
function string(rule) {
    return function() {
        if (input[position] === rule) {
            return {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position += rule.length
            }
        } else {
            return false;
        }
    }
}
```

```
function wrapper(parsers) {
    return function() {
        var asts = parsers.map(function(parser) {
            return parser();
        })
        if (asts.indexOf(false) < 0) {
            return {
                type: 'wrapper',
                match: null,
                children: asts
            }
        } else {
            return false;
        }
    }
}
```

```
function wrapper(parsers) {
    return function() {
        var asts = parsers.map(function(parser) {
            return parser();
        });
        if (asts.indexOf(false) < 0) {
            return {
                type: 'wrapper',
                match: null,
                children: asts
            }
        } else {
            return false;
        }
    }
}
```

```
function wrapper(parsers) {
    return function() {
        var asts = parsers.map(function(parser) {
            return parser();
        });
        if (asts.indexOf(false) < 0) {
            return {
                type: 'wrapper',
                match: null,
                children: asts
            }
        } else {
            return false;
        }
    }
}
```

```
function wrapper(parsers) {
    return function() {
        var asts = parsers.map(function(parser) {
            return parser();
        });
        if (asts.indexOf(false) < 0) {
            return {
                type: 'wrapper',
                match: null,
                children: asts
            }
        } else {
            return false;
        }
    }
}
```

```
function wrapper(parsers) {
    return function() {
        var asts = parsers.map(function(parser) {
            return parser();
        });
        if (asts.indexOf(false) < 0) {
            return {
                type: 'wrapper',
                match: null,
                children: asts
            }
        } else {
            return false;
        }
    }
}
```

```
var parser = wrapper([
    string('a')
]);

var ast = parser();  // input = 'a'
```

```
var parser = wrapper([
    string('a')
]);

var ast = parser();  // input = 'a'
```

```
{
  "type": "wrapper",
  "match": null,
  "children": [
    {
      "type": "string",
      "match": "a",
      "start_position": 0,
      "end_position": 1
    }
  ]
}
```

Грамматика и Абстрактная Формальная Грамматика

```
// это методы парсера

function string(parsers) {
    ..
}

function wrapper(parsers) {
    ..
}
```

```
// это грамматика

var parser = wrapper([
    string('a')
]);
```

?

PEG

(Грамматика Парсерщая Выражения)

// Rule
// Parsing Expression

// Parsing Expression
// Sequence: e1 e2 e3
// Ordered choice: e1 / e2
// Zero-or-more: e*
// One-or-more: e+
// Optional: e?
// And-predicate: &e
// Not-predicate: !e

// Atomic Parsing Expression:
// String
// RegExp
// Rule

rule_name -> parsing_expression;

// Rule
// Parsing Expression

// Parsing Expression
// Sequence: e1 e2 e3
// Ordered choice: e1 / e2
// Zero-or-more: e*
// One-or-more: e+
// Optional: e?
// And-predicate: &e
// Not-predicate: !e

// Atomic Parsing Expression:
// String
// RegExp
// Rule

e1 e2 e3

// Rule
// Parsing Expression

// Parsing Expression
// Sequence: e1 e2 e3
// Ordered choice: e1 / e2
// Zero-or-more: e*
// One-or-more: e+
// Optional: e?
// And-predicate: &e
// Not-predicate: !e

// Atomic Parsing Expression:
// String
// RegExp
// Rule

e1 / e2 / e3

```
// Rule
//   Parsing Expression

// Parsing Expression
//   Sequence: e1 e2 e3
//   Ordered choice: e1 / e2
//   Zero-or-more: e*
//   One-or-more: e+
//   Optional: e?
//   And-predicate: &e
//   Not-predicate: !e

// Atomic Parsing Expression:
//   String
//   RegExp
//   Rule
```

e*

```
// Rule
//     Parsing Expression

// Parsing Expression
//     Sequence: e1 e2 e3
//     Ordered choice: e1 / e2
//     Zero-or-more: e*
//     One-or-more: e+
//     Optional: e?
//     And-predicate: &e
//     Not-predicate: !e

// Atomic Parsing Expression:
//     String
//     RegExp
//     Rule
```

e+

// Rule
// Parsing Expression

// Parsing Expression
// Sequence: e1 e2 e3
// Ordered choice: e1 / e2
// Zero-or-more: e*
// One-or-more: e+
// Optional: e?
// And-predicate: &e
// Not-predicate: !e

// Atomic Parsing Expression:
// String
// RegExp
// Rule

e?

// Rule
// Parsing Expression

// Parsing Expression
// Sequence: e1 e2 e3
// Ordered choice: e1 / e2
// Zero-or-more: e*
// One-or-more: e+
// Optional: e?
// And-predicate: &e
// Not-predicate: !e

// Atomic Parsing Expression:
// String
// RegExp
// Rule

&e

// Rule
// Parsing Expression

// Parsing Expression
// Sequence: e1 e2 e3
// Ordered choice: e1 / e2
// Zero-or-more: e*
// One-or-more: e+
// Optional: e?
// And-predicate: &e
// Not-predicate: !e

// Atomic Parsing Expression:
// String
// RegExp
// Rule

!e

```
// Rule
//     Parsing Expression

// Parsing Expression
//     Sequence: e1 e2 e3
//     Ordered choice: e1 / e2
//     Zero-or-more: e*
//     One-or-more: e+
//     Optional: e?
//     And-predicate: &e
//     Not-predicate: !e

// Atomic Parsing Expression:
//     String
//     RegExp
//     Rule
```

```
// Rule
//     Parsing Expression

// Parsing Expression
//     Sequence: e1 e2 e3
//     Ordered choice: e1 / e2
//     Zero-or-more: e*
//     One-or-more: e+
//     Optional: e?
//     And-predicate: &e
//     Not-predicate: !e

// Atomic Parsing Expression:
//     String
//     RegExp
//     Rule
```

[0-9] [a-z]

[\w] [\s]

. (точка - любой символ)

```
// Rule
//     Parsing Expression

// Parsing Expression
//     Sequence: e1 e2 e3
//     Ordered choice: e1 / e2
//     Zero-or-more: e*
//     One-or-more: e+
//     Optional: e?
//     And-predicate: &e
//     Not-predicate: !e

// Atomic Parsing Expression:
//     String
//     RegExp
//     Rule
```

rule_name

```
function string(rule) {
    return function() {
        lastExpectations = [];
        if (input.indexOf(rule, position) === position) {
            var node = {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position + rule.length
            };
            position = node.end_position;
            return node;
        } else {
            lastExpectations = [{
                type: 'string',
                rule: rule,
                position: position
            }]
            return false;
        }
    }
}
```

```
function string(rule) {
    return function() {
        lastExpectations = [];
        if (input.indexOf(rule, position) === position) {
            var node = {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position + rule.length
            };
            position = node.end_position;
            return node;
        } else {
            lastExpectations = [{
                type: 'string',
                rule: rule,
                position: position
            }]
            return false;
        }
    }
}
```

```
function string(rule) {
    return function() {
        lastExpectations = [];
        if (input.indexOf(rule, position) === position) {
            var node = {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position + rule.length
            };
            position = node.end_position;
            return node;
        } else {
            lastExpectations = [{
                type: 'string',
                rule: rule,
                position: position
            }]
            return false;
        }
    }
}
```

```
function string(rule) {
    return function() {
        lastExpectations = [];
        if (input.indexOf(rule, position) === position) {
            var node = {
                type: 'string',
                match: rule,
                start_position: position,
                end_position: position + rule.length
            };
            position = node.end_position;
            return node;
        } else {
            lastExpectations = [{
                type: 'string',
                rule: rule,
                position: position
            }]
            return false;
        }
    }
}
```

```
function sequence(parsing_expressions) {
    return function() {
        var nodes = [];
        var start_position = position;
        for (var i = 0; i < parsing_expressions.length; i++) {
            var node = parsing_expressions[i]();
            if (node) {
                nodes.push(node);
            } else {
                return false;
            }
        }
        var match = nodes.reduce(function(r, n){ return r +
(n.match || '') }, '');
        return {
            type: 'sequence',
            match: match,
            children: nodes,
            start_position: start_position,
            end_position: position
        };
    }
}
```

```
function sequence(parsing_expressions) {
    return function() {
        var nodes = [];
        var start_position = position;
        for (var i = 0; i < parsing_expressions.length; i++) {
            var node = parsing_expressions[i]();
            if (node) {
                nodes.push(node);
            } else {
                return false;
            }
        }
        var match = nodes.reduce(function(r, n){ return r +
(n.match || '') }, '');
        return {
            type: 'sequence',
            match: match,
            children: nodes,
            start_position: start_position,
            end_position: position
        };
    }
}
```

```
function sequence(parsing_expressions) {
    return function() {
        var nodes = [];
        var start_position = position;
        for (var i = 0; i < parsing_expressions.length; i++) {
            var node = parsing_expressions[i]();
            if (node) {
                nodes.push(node);
            } else {
                return false;
            }
        }
        var match = nodes.reduce(function(r, n){ return r +
(n.match || '') }, '');
        return {
            type: 'sequence',
            match: match,
            children: nodes,
            start_position: start_position,
            end_position: position
        };
    }
}
```

```
function sequence(parsing_expressions) {
    return function() {
        var nodes = [];
        var start_position = position;
        for (var i = 0; i < parsing_expressions.length; i++) {
            var node = parsing_expressions[i]();
            if (node) {
                nodes.push(node);
            } else {
                return false;
            }
        }
        var match = nodes.reduce(function(r, n){ return r +
(n.match || '') }, '');
        return {
            type: 'sequence',
            match: match,
            children: nodes,
            start_position: start_position,
            end_position: position
        };
    }
}
```

```
rd.setState({
    input: 'AB',
    position: 0
})
// act
var parser = rd.ordered_choice([
    rd.sequence([
        rd.string('A'),
        rd.string('A')
    ]),
    rd.string('B')
]);
var ast = parser();
```

```
rd.setState({
    input: 'AB',
    position: 0
})
// act
var parser = rd.ordered_choice([
    rd.sequence([
        rd.string('A'),
        rd.string('A')
    ]),
    rd.string('B')
]);
var ast = parser();
```

```
Unexpected "B"
1: AB
----^
```

```
rd.setState({
    input: 'AA',
    position: 0
})
// act
var parser = rd.ordered_choice([
    rd.sequence([
        rd.string('A'),
        rd.string('A')
    ]),
    rd.string('B')
]);
var ast = parser();
```

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

Система описания синтаксиса

BNF

Backus-Naur Form

BNF

```
<syntax>         ::= <rule> | <rule> <syntax>
<rule>           ::= <opt-whitespace> "<" <rule-name> ">" <
<opt-whitespace> ::= " " <opt-whitespace> | ""
<expression>     ::= <list> | <list> <opt-whitespace> "|" <
<line-end>       ::= <opt-whitespace> <EOL> | <line-end> <l
<list>           ::= <term> | <term> <opt-whitespace> <list
<term>           ::= <literal> | "<" <rule-name> ">"
<literal>        ::= '"' <text> '"' | "'" <text> "'"
```

BNF -> **EBNF**, ABNF ...

```
GRAMMAR test

next_seq -> sequenceA+ orderedB+;

sequenceA -> "begin"* [0-9] "end";
orderedB -> "A" / [0-9]+;
```

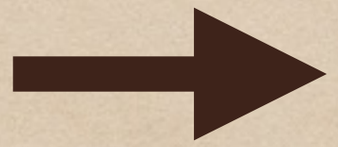
```
GRAMMAR url

url         ->  scheme "://" host pathname search hash?;
scheme      ->  "http" "s"?;
host        ->  hostname port?;
hostname    ->  segment ("." segment)*;
segment     ->  [a-z0-9-]+;
port        ->  ":" [0-9]+;
pathname    ->  "/" [^ ?]*;
search      ->  ("?" [^ #]*)?;
hash        ->  "#" [^ ]*;
```

VISITOR

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

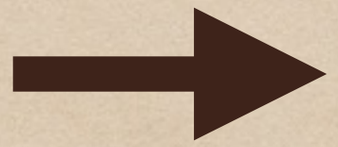

```
[
  [
    "A",
    "A"
  ]
]
```

```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

```
var parser = rd.ordered_choice([
    rd.sequence([
        rd.string('A'),
        rd.string('A')
    ]),
    rd.string('B')
]);
var ast = parser();
```

```
var parser = rd.ordered_choice([
    rd.sequence([
        rd.string('A'),
        rd.string('A')
    ]),
    rd.string('B')
]);
var ast = parser();
```



```
{
  "type": "ordered_choice",
  "match": "AA",
  "children": [
    {
      "type": "sequence",
      "match": "AA",
      "children": [
        {
          "type": "string",
          "match": "A",
          "start_position": 0,
          "end_position": 1
        },
        {
          "type": "string",
          "match": "A",
          "start_position": 1,
          "end_position": 2
        }
      ],
      "start_position": 0,
      "end_position": 2
    }
  ],
  "start_position": 0,
  "end_position": 2
}
```

Парсер PEG грамматики методом рекурсивного спуска, с использованием рекурсивного каррирования и композиций функций, с EBNF подобной системой описания синтаксиса

https://github.com/aiboy/SPEG

Что дальше?

aiboy / **SPEG**

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Wiki | Pulse | Graphs | Settings

Simple PEG (parsing expression grammar) for JavaScript and Python — Edit

19 commits | **1** branch | **0** releases | **1** contributor

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

aiboy finished with SPEG_action_visitor and SPEG on python part — Latest commit `1ce2e25` 19 hours ago

js_test	SPEG pytho evaluation (string and regexp) for now	a day ago
py_test	finished with SPEG_action_visitor and SPEG on python part	19 hours ago
speg_fixtures	finished with SPEG_action_visitor and SPEG on python part	19 hours ago
speg_grammar_fixtures	SPEG pytho evaluation (string and regexp) for now	a day ago
.gitignore	SPEG pytho evaluation (string and regexp) for now	a day ago
LICENCE	fix licence year	3 days ago
README.md	Initial commit	26 days ago
demo-peg.js	initial commit	26 days ago
demo-rdp.js	initial commit	26 days ago
index.js	Small change returned back	7 days ago
index.py	finished with SPEG_action_visitor and SPEG on python part	19 hours ago
package.json	finished with SPEG_action_visitor and SPEG on python part	19 hours ago

PEG.js

Parser Generator for JavaScript

PEG.js is a simple parser generator for JavaScript that produces fast parsers with excellent error reporting. You can use it to process complex data or computer languages and build transformers, interpreters, compilers and other tools easily.

Features

- Simple and expressive grammar syntax
- Integrates both lexical and syntactical analysis
- Parsers have excellent error reporting out of the box
- Based on parsing expression grammar formalism — more powerful than traditional LL(*k*) and LR(*k*) parsers
- Usable from your browser, from the command line, or via JavaScript API

Try PEG.js online

— or —

```
npm install pegjs
```

— or —

```
bower install pegjs
```

— or —

Download browser version

- **PEG.js — minified**
- **PEG.js — development**

Canopy, a parser compiler

Canopy is a **PEG** parser compiler. It lets you describe the grammar of the language you're trying to parse using a simple, terse syntax, and it generates a parser for the language from this definition.

You can install the command-line tools through `npm`:

```
$ npm install -g canopy
```

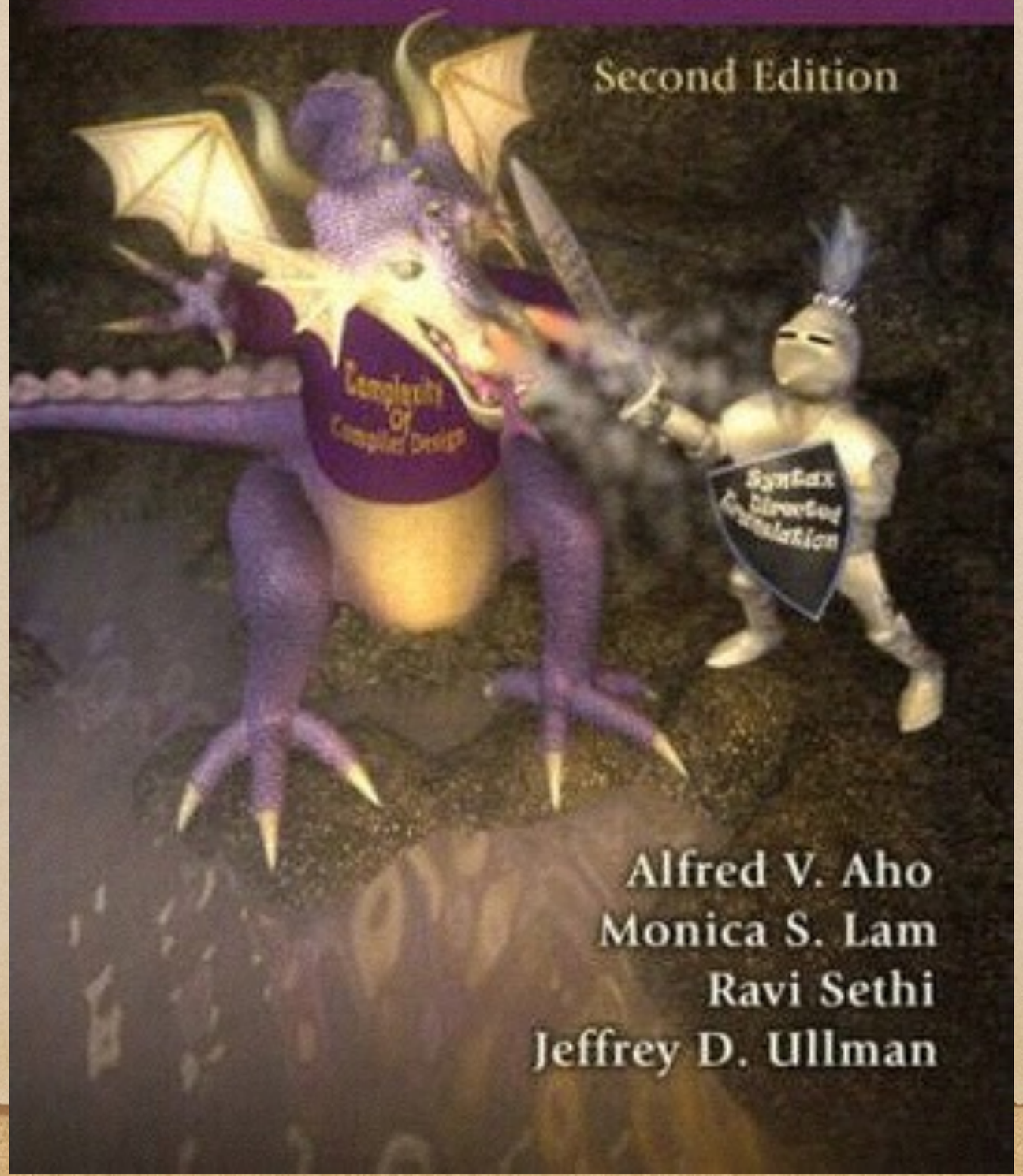
Canopy can generate parsers in the following languages:

- **Java**
- **JavaScript**
- **Python**
- **Ruby**

Compilers

Principles, Techniques, & Tools

Second Edition



Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

Calculator demo

This demo parses mathematical expressions and returns the answer, keeping the correct order of operations.

Enter an expression to evaluate, such as PI*4^2 + 5:

PI*4^2 + 5 equals

The grammar

This Jison grammar was used to create the parser/evaluator:

```
/* description: Parses end evaluates mathematical expressions. */

/* lexical grammar */
%lex
%%
\s+                   {/* skip whitespace */}
[0-9]+("."[0-9]+)?\b  {return 'NUMBER';}
```

ternjs / acorn

Watch 73 | Unstar 1,664 | Fork 201

Code | Issues 11 | Pull requests 1 | Wiki | Pulse | Graphs

A small, fast, JavaScript-based JavaScript parser

706 commits | 1 branch | 36 releases | 51 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

TimothyGu committed with marijnh CHANGELOG: Fix date — Latest commit 6618b5b a day ago

bin	Speed up generate-identifier-regex	4 months ago
dist	Use regexps instead of magic generated functions	8 months ago
src	[loose parser] Make sure ExportAllDeclaration always has a source node	19 days ago
test	Disallow shorthand properties with keyword names	a month ago
.editorconfig	Editorconfig: enforce Unix line endings and extra new line in the end…	2 years ago
.gitattributes	Force LF endings in code.	2 years ago
.gitignore	Add bin/acorn to .gitignore	8 months ago
.npmignore	Make sure all ignored files are ignored in npmigore	a year ago
.tern-project	[.tern-project] Load node and es_modules plugins	10 months ago
.travis.yml	[travis.yml] Add sudo: false	9 months ago
AUTHORS	Mark release 3.1.0	2 months ago
CHANGELOG.md	CHANGELOG: Fix date	17 hours ago
LICENSE	Update license year range to 2016	5 months ago
README.md	Add few more plugins	3 days ago
package.json	Mark release 3.1.0	2 months ago

ECMAScript parsing infrastructure for multipurpose analysis

Esprima is a high performance, standard-compliant ECMAScript parser written in ECMAScript (also popularly known as JavaScript).

Features

- Full support for ECMAScript 6 (ECMA-262)
- Sensible syntax tree format as standardized by EStree project
- Optional tracking of syntax node location (index-based and line-column)
- Heavily tested (~1200 tests with full code coverage)

Esprima serves as an important **building block** for some JavaScript language tools, from code instrumentation to editor autocompletion.

Esprima

```
1  var capitalDb = {
2      Indonesia: 'Jakarta',
3      Germany: 'Berlin',
4      Norway: 'Oslo'
5  };
6
7  // Property completion: "capitalDb." and press Ctrl+Space.
8  capitalDb.
9
```

```
Germany : String
Indonesia : String
Norway : String
---
hasOwnProperty(property) : Boolean
isPrototypeOf(object) : Boolean
propertyIsEnumerable(property) : Boolean
toLocaleString() : String
toString() : String
```

Once the full syntax tree is obtained, various **static code analysis** can be applied to give an insight to the code: syntax visualization, code validation, editing autocomplete (with type inferencing) and many others.

Regenerating the code from the syntax tree permits a few different types of **code transformation**, from a simple rewriting (with specific formatting) to a more complicated minification.

Esprima runs on many popular web browsers, as well as other ECMAScript platforms such as Rhino, Nashorn, and Node.js. It is distributed under the BSD license.

reworkcss / css

CSS parser / stringifier for Node.js

138 commits **3** branches **20** releases **18** contributors

Branch: **master** New pull request Create new file Upload files Find file Clone or download

kevva Merge pull request #86 from dominicbarnes/master ... Latest commit `0f5ad51` on 6 Jan

benchmark	Add css-parse benchmarks	2 years ago
lib	include optional source on returned stylesheet object (fixes #85)	5 months ago
test	include optional source on returned stylesheet object (fixes #85)	5 months ago
.gitignore	ignore node_modules	4 years ago
.travis.yml	Create .travis.yml	3 years ago
History.md	Update History.md	a year ago
LICENSE	Create LICENSE	3 years ago
Readme.md	fix typo	a year ago
generate-tests.js	Add test case generator script	2 years ago
index.js	Add css-parse and css-stringify modules	2 years ago
package.json	2.2.1	a year ago

Readme.md

Syntaxation • Douglas Crockford

GOTO Conferences

Подписаться 42 403

8 119 просмотров

Добавить в Поделиться ••• Ещё

121 2

Роман Дворнов

avito

CSSO - оптимизируем CSS


THE
END

Q&A?

Алексей Охрименко

http://bit.ly/1U1Fpa8

Twitter: @Ai_boy