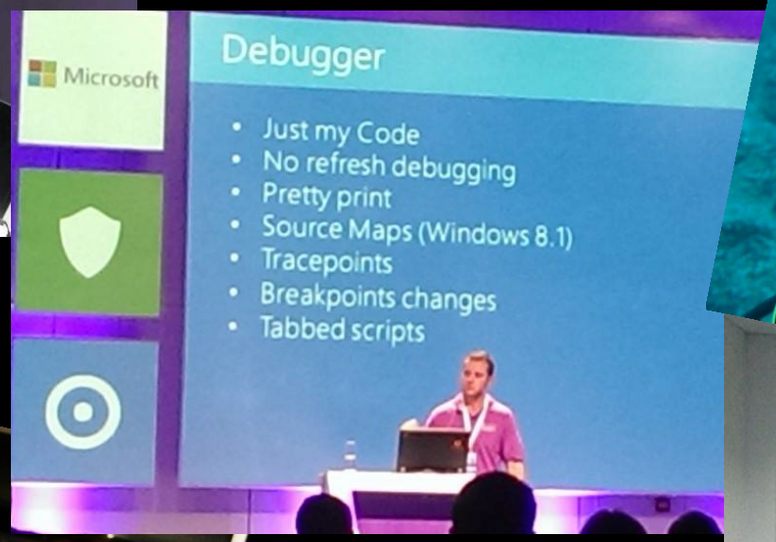
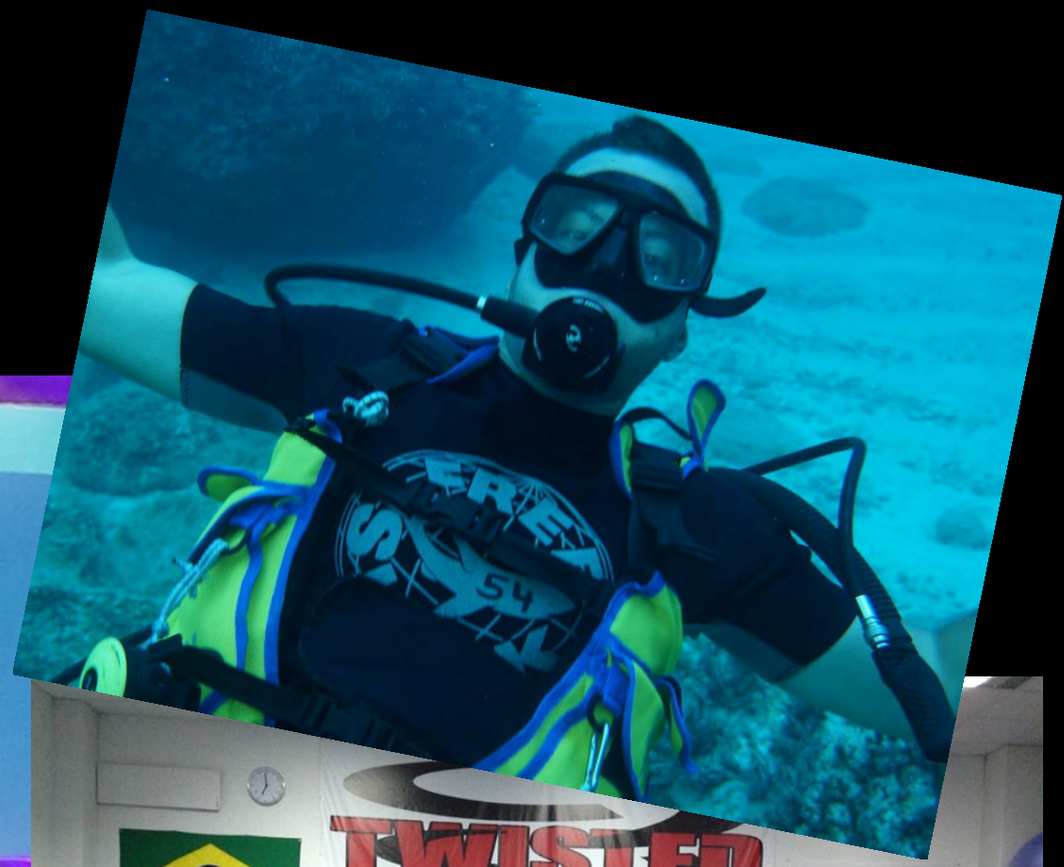
JavaScript Performance Analysis and Testing by Instrumentation

Angel Todorov
Principal Architect, Infragistics Inc.

About me

github.com/attodorov





JavaScript Evolution

alert(“pwned”) → element.style → $ → Node.JS

Topics

Subscribe

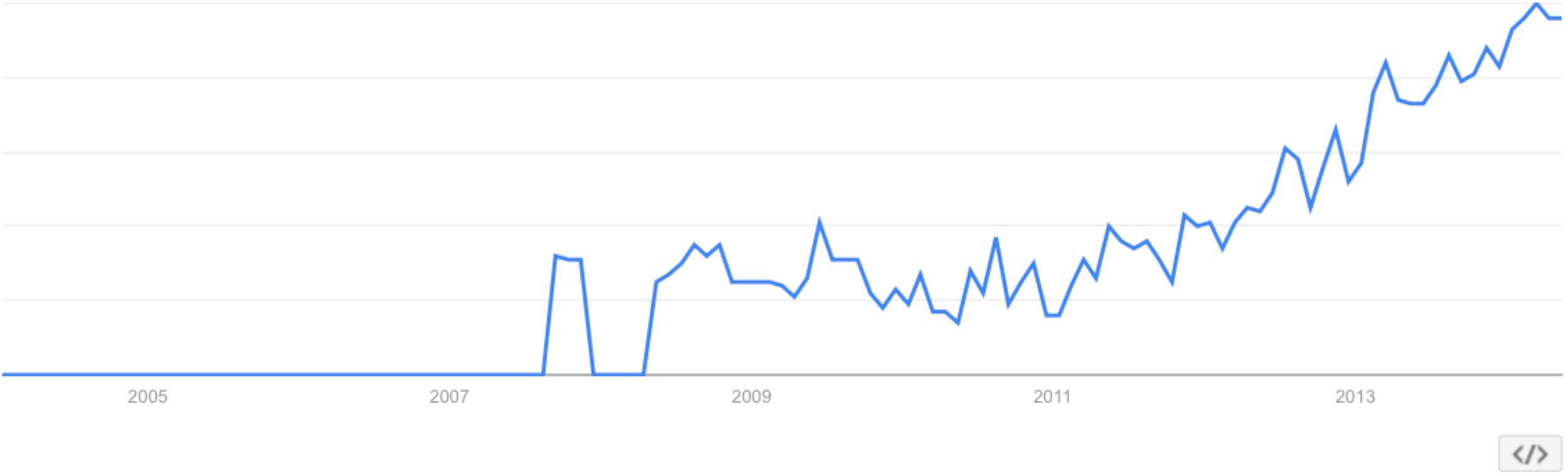
js performance
Search term

+ Add term

Interest over time

News headlines

Forecast



Google found that a half-second increase in the time it took the search engine to display results caused a *20% drop* in traffic among test participants.

Amazon reportedly found that a 100-millisecond delay in the time it takes their pages to load could lead to a 1% drop in revenue.

$74.5B…


HEAT
ForGIFs.com

75% of people would not return to websites that took longer than 4 seconds to load.

The Idea

- I want to automate performance analysis of **complete web apps**
- Similar to Firebug / Chrome
- should work for any platform - Node, browsers
- 100 % unattended, OS agnostic, unobtrusive
- easy & flexible reporting

And answer questions like…

- How long did it take to render my list of Products for build XYZ
- Was it slower than build XYZ - 1?
- Which functions caused the slowdown(s)?
- What % was it slower/faster ?
- Which functions are invoked most?
- Does setting a width/height affect performance? By how much?

The State of JS

Performance Testing

(web) app performance testing

vs

Unit performance testing

1

Browser tooling

jQuery Controls – Samples

igniteui.com

IGNITEUI™
INFRAGISTICS JQUERY CONTROLS

- Application Samples
- Getting Started
- Configurator
- jQuery Controls

Menu

Download Now

jQuery Controls

Responsive web design on any browser, any platform and any device

Data Grids

Financial Analysis & Stock Tracking
Launch Download

Elements Resources Network Sources Timeline Profiles Audits Console Ember

Profiles

CPU PROFILES

Profile 1

HEAP SNAPSHOTS

Snapshot 1
4.9 MB

Snapshot 2
4.9 MB

HEAP TIMELINES

Class filter

Constructor	Distance	Objects Count	Shallow Size	Retained Size
▶ $	2	1 0%	80 0%	6 996 0%
▶ (array)	1	22 074 22%	493 796 29%	1 909 008 37%
▼ (closure)	1	10 916 11%	392 976 8%	1 778 116 35%
▶ function getSelection() @1799	2		36 0%	128 0%
▶ function Empty() @1801	1		36 0%	36 0%

Object's retaining tree

Object	Dista...	Shallow S...	Retained ...
▶ $ in Window / igniteui.com/ @10463	1	40 0%	59 296 7%
▶ jQuery in Window / igniteui.com/ @10463	1	40 0%	59 296 7%
▶ constructor in @32499	3	92 0%	1 964 0%
▶ b in system / Context @32639	3	656 0%	48 176 3%

Summary ▼ All objects ▼ ⚠1

IG Finance

igniteui.com/igfinance-sample/

Google

FINANCE

MSFT 40.19

Microsoft Corpora (MSFT)

40.19 ↑0.07 (0.17%)

5/28/2014 0:31:39

Range	39.81 -		40.13
52 Weeks	30.84 -		40.10
Open	40.22	y High	40.26
Volume	261604	y Low	39.81

AREA | CANDLESTICK | OHLC | LINE

SEP OCT NOV DEC JAN FEB

Indicator/Period: Money Flow Index 14

100 80 60

Console | Inspector | Debugger | Style Editor | Profiler | Network

Import...

Profile 1
Completed

Profile 2
Completed Save

Complete Profile

Running Time		Self	
5950	100.0%	2251	▼ (root)
2390	40.2%	2	▶ ret() @ infragistics.core.js:208 igniteui.com
526	8.8%	1	▶ makeInfallible/<() @ DevToolsUtils.js:78 gre
406	6.8%	25	▶ fetch/streamListener.onStopRequest() @ script.js:4971 gre
321	5.4%	0	▶ b.event.add/v.handle() @ jquery-1.9.1.min.js:3 igniteui.com
14	0.2%	1	▶ TM_notify() @ nsUpdateTimerManager.js:128 gre
11	0.2%	11	MousePosTracker.handleEvent() @ browser.js:16820 browser
8	0.1%	0	▶ fetch/streamListener.onDataAvailable() @ script.js:4968 gre
6	0.1%	1	▶ WalkerActor<.onMutations() @ inspector.js:1926 gre
5	0.1%	0	▶ $.ig.util.setSize/obj.onTick</obj.tickID<() @ infragistics.core.js:208 igniteui.com

http://igniteui.com/world-stats-sample/

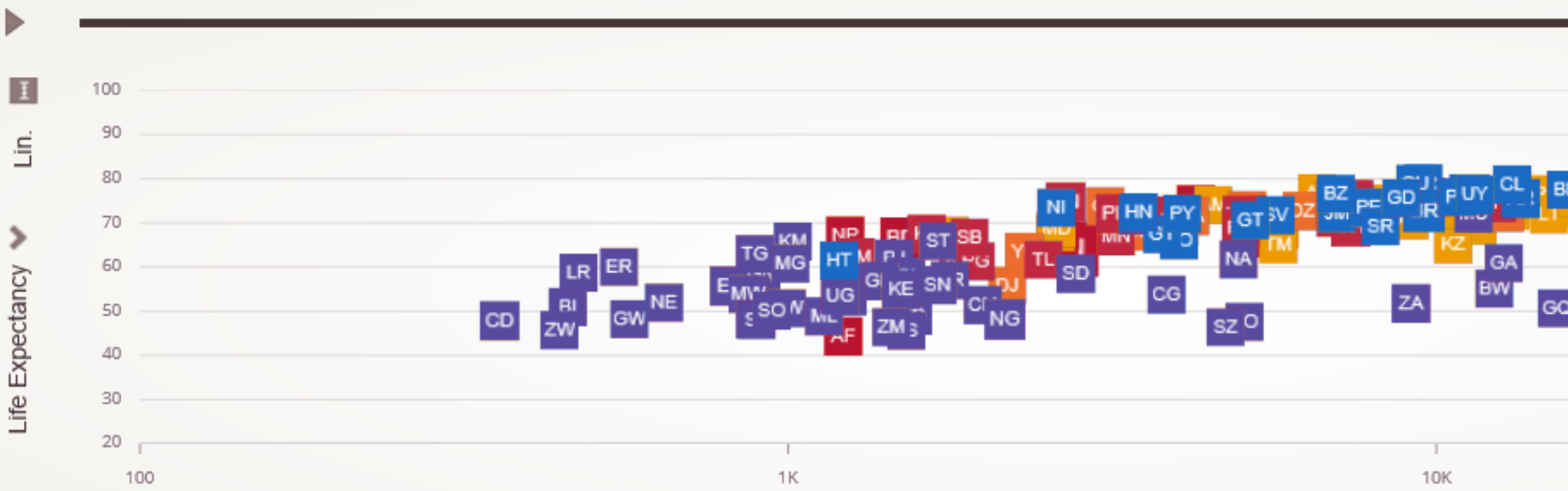
WorldStats

worldstats

Limit Countries to View

select up to four to compare

- Armenia
- Aruba
- Australia
- Austria
- Azerbaijan
- Bahamas
- Bahrain
- Bangladesh



F12

UI Responsiveness

Diagnostic session: 7.505 seconds

1s 2s 3s 4s 5s

CPU utilization (%)

200
100

Visual throughput (FPS)

60

Category: Scripting

2

Perf. Unit testing libraries / apps

Benchmark.js v1.0.0

A robust benchmarking library that works on nearly all JavaScript platforms, supports high-resolution timers, and returns statistically significant results.

```
var suite = new Benchmark.Suite;

// add tests
suite.add('RegExp#test', function() {
  /o/.test('Hello World!');
})
.add('String#indexOf', function() {
  'Hello World!'.indexOf('o') > -1;
})
.add('String#match', function() {
  !!'Hello World!'.match(/o/);
})
// add listeners
.on('cycle', function(event) {
  console.log(String(event.target));
})
.on('complete', function() {
  console.log('Fastest is ' + this.filter('fastest').pluck('name'));
})
```

jsPerf — JavaScript performance playground

What is jsPerf?

jsPerf aims to provide an easy way to create and share test cases, comparing the performance of different JavaScript snippets by running benchmarks. For more information, see the FAQ.

Create a test case

Your details (optional)

Name

Email (won't be displayed; might be used for Gravatar)

URL

Test case details

Title *

Slug *

Test case URL will be `http://jsperf.com/slug`

Published (uncheck if you want to fiddle around before making the page public)

Description

3. Dynamic prototypical injection

```
function bootstrap(item, path) {
  var original = item[prop];
  if (item[prop] instanceof Function) {
    // Infect the actual function
    infect(item, prop, path);
    // Infect the functions prototype
    if (item[prop].prototype) {
      bootstrap(item[prop], 'prototype', path
+'.prototype');
    }
```

PUBLIC holidayextras / hxTracer

A dependency-free Javascript Tracer

15 commits | **5** branches | **0** releases | **6** contributors

branch: **master** hxTracer / +

Removing old artifact

hxoliverrumbelow authored on Apr 9 latest commit `f928eb6e00`

COPYING.txt	initial commit	5 months ago
README.md	Fix quote typo	3 months ago
index.js	my ide did not like your semicolonage	3 months ago
package.json	Adding exception for util.format, fixed bug with reporting	4 months ago
test.js	my ide did not like your semicolonage	3 months ago

HTTPS clone URL

https://github.com/h

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

README.md

Introduction

I wrote this javascript tracer to gain a better insight into how a project worked. Hopefully it'll help others too. Keep reading for an explanation on how it works.

4.

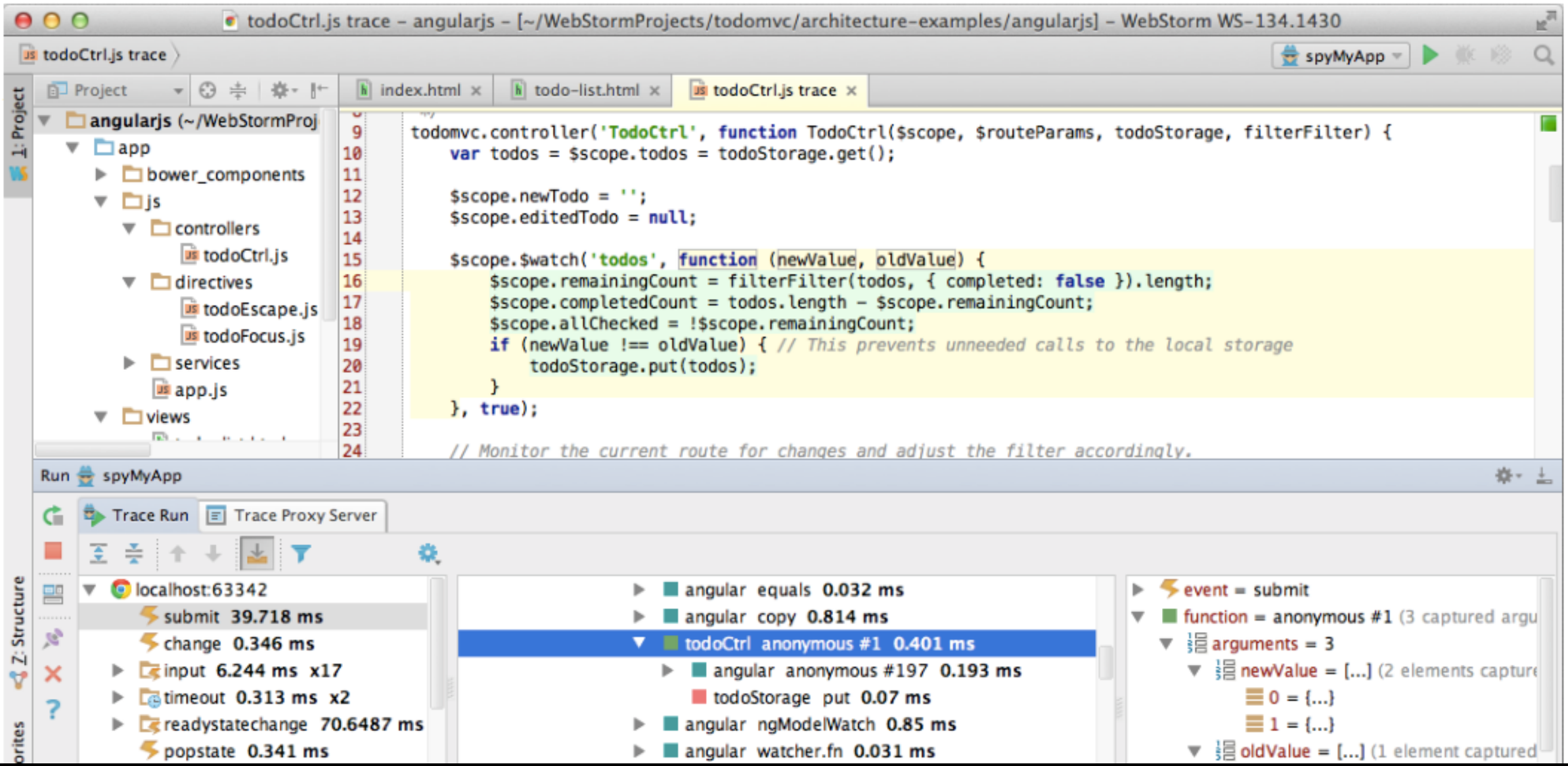
Instrumentation

The ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information.

Can be of source or binary type

javascript tracing made simple

Check out WebStorm 8 integration: lots of new features and fixed bugs, significant performance and usability improvements



So why bother?

Not automation / CI friendly

Not transparent

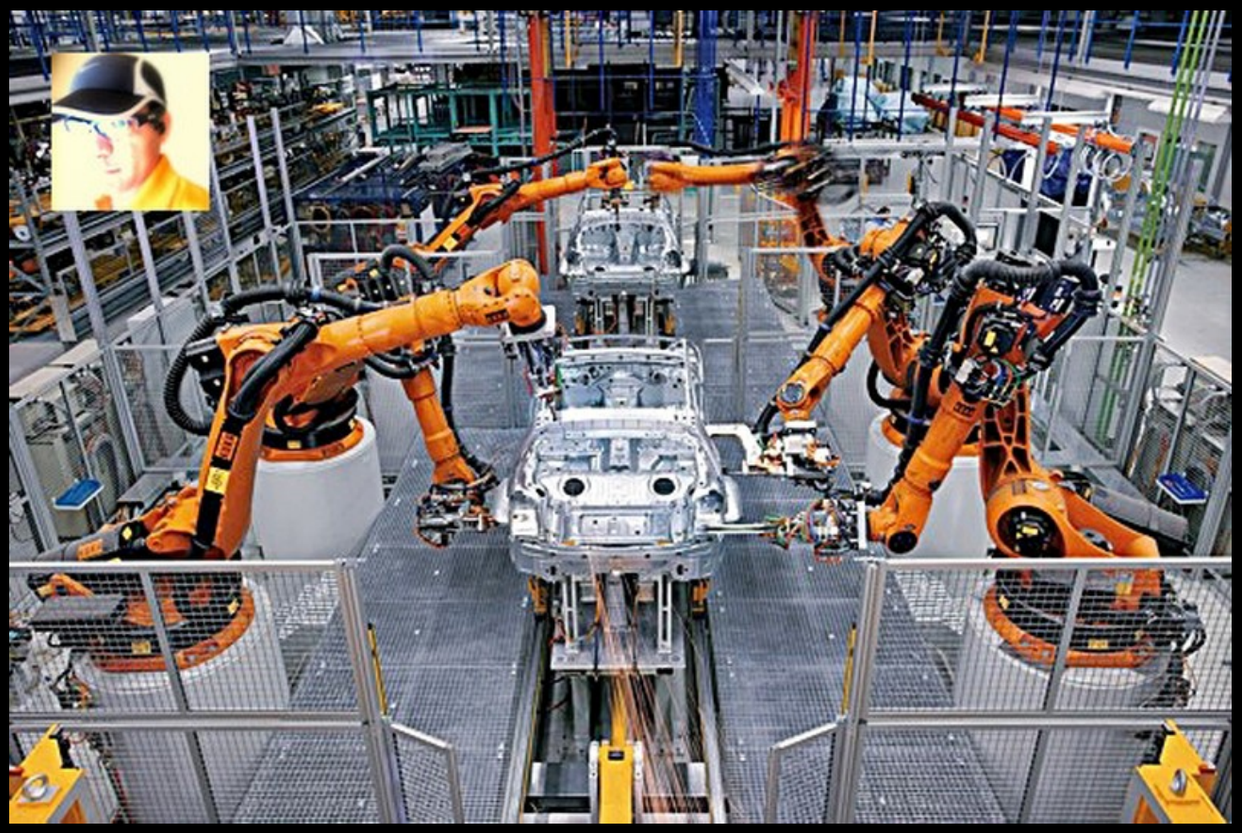
Not platform agnostic

Lack of extensibility hooks, **extra semantics** & generic reporting

Not Open Source

General level of automation

Functional Testing



Performance Testing

SYSTEM FAILURE

How are compilers made?

Behind every (successful) (programming) language, there is a grammar that defines its structure

Anonymous

Programming languages are described by context-free grammars

a set of recursive rewriting rules (*productions*) used to generate patterns of strings.

1. <expression> --> number
2. <expression> --> (<expression>)
3. <expression> --> <expression> + <expression>
4. <expression> --> <expression> - <expression>
5. <expression> --> <expression> * <expression>
6. <expression> --> <expression> / <expression>

Grammars are written in EBNF (Extended Backus-Naur) format

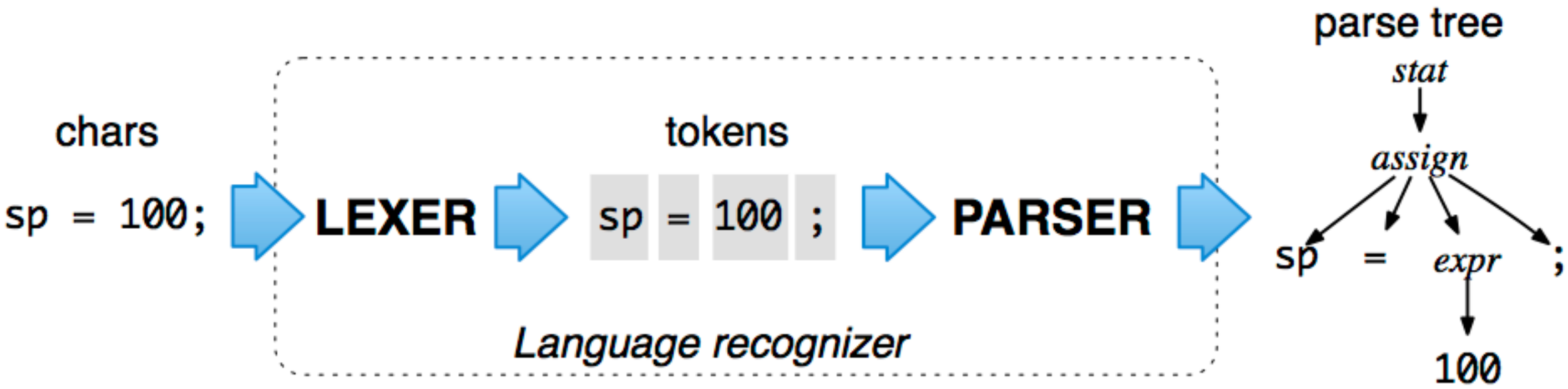
IfStatement ::= "if"
"(" Expression ")" Statement ("else" Statement)?

ClassDeclaration= "class", Identifier, ["extends", Identifier],
"{", { VarDeclaration }, { MethodDeclaration } "}";

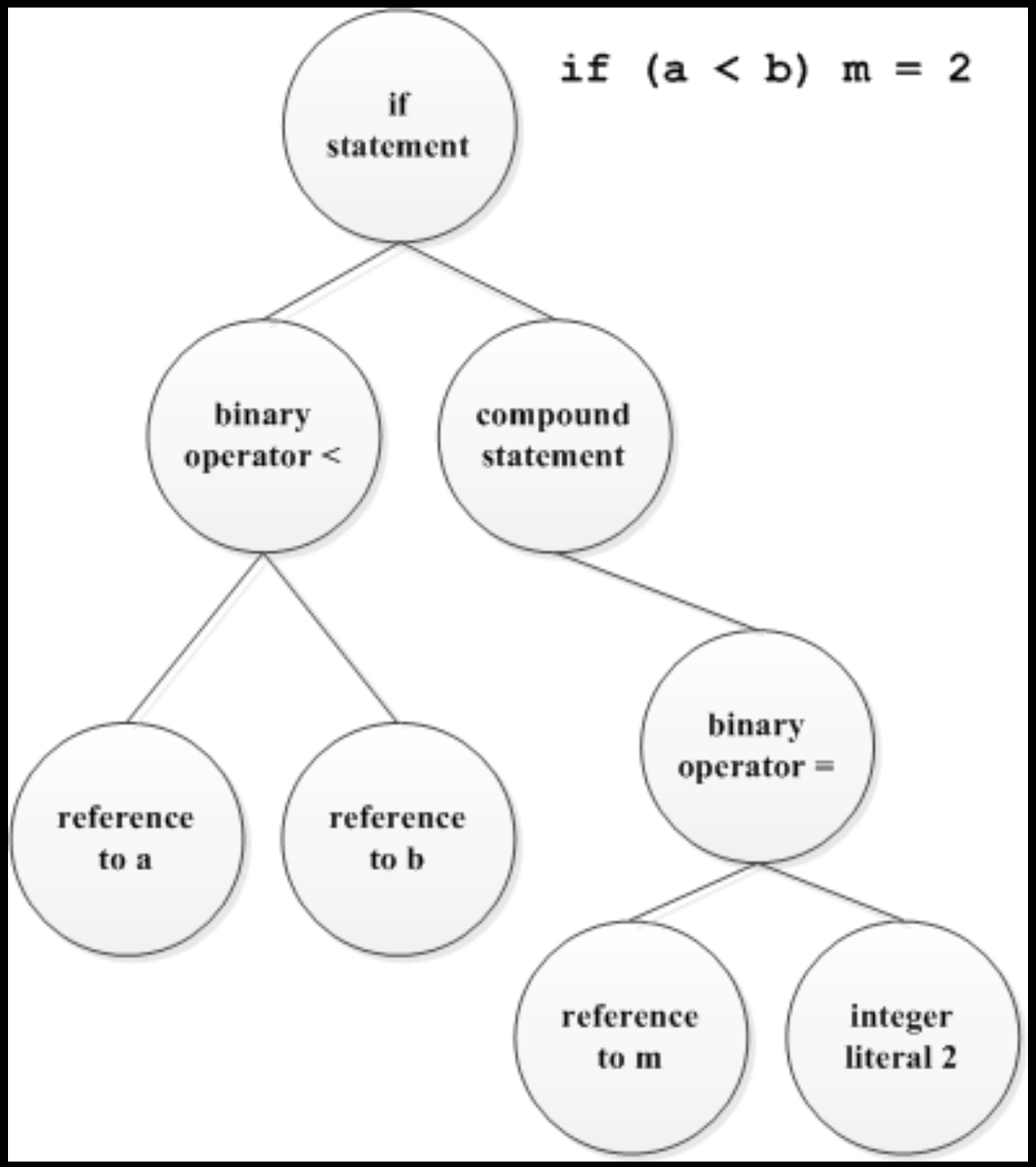
ecmascriptA3.g

```
1024
1025  emptyStatement:
1026      ;
1027
1028  expressionStatement:
1029      /* [lookahead not a member of {{, function}} */ expression SEMI
1030      ;
1031
1032  ifStatement:
1033      'if' LPAREN expression RPAREN statement 'else' statement
1034      | 'if' LPAREN expression RPAREN statement
1035      ;
1036
1037  iterationStatement:
1038      'do' statement 'while' LPAREN expression RPAREN SEMI
1039      | 'while' LPAREN expression RPAREN statement
1040      | 'for' LPAREN (
1041          (expressionNoln)? SEMI (expression)? SEMI (expression)? RPAREN statement
1042          | 'var' variableDeclarationListNoln SEMI (expression)? SEMI (expression)?
1043          | leftHandSideExpression 'in' expression RPAREN statement
1044          | 'var' variableDeclarationNoln 'in' expression RPAREN statement
1045          )
1046      ;
1047
1048  continueStatement:
1049      'continue' /* [ no line terminator here ] */ (identifier)? SEMI
1050      ;
1051
1052  breakStatement:
1053      'break' /* [ no line terminator here ] */ (identifier)? SEMI
1054      ;
```

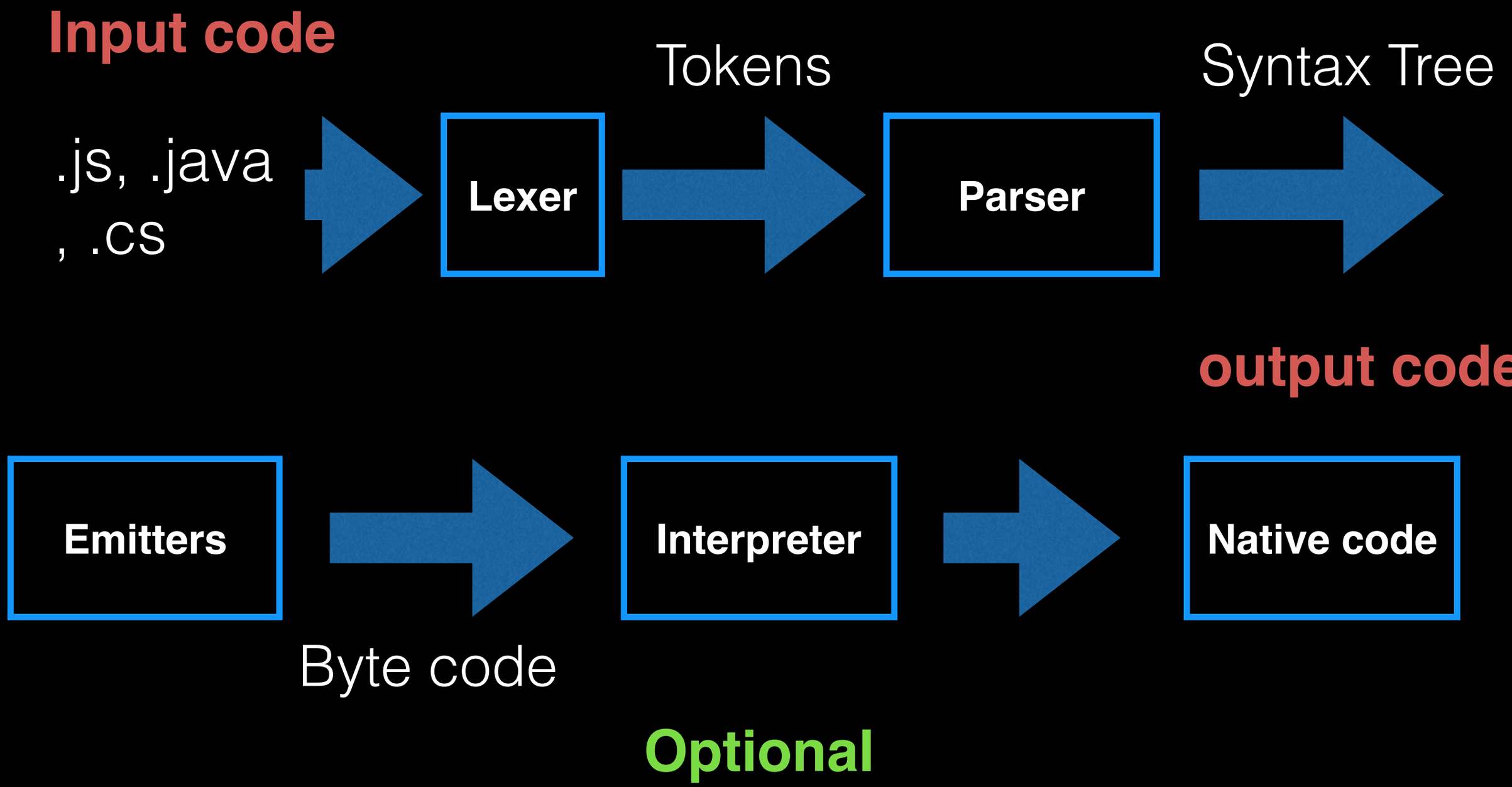
We generate **lexers** and **parsers** from EBNF grammars



AST Example


if (a < b) m = 2
if statement
binary operator <
compound statement
reference to a
reference to b
binary operator =
reference to m
integer literal 2

A Simple Compiler Architecture



input code + modifications
≡ instrumented input code
(output)

Instrumented function example

```
function doWork() {
  var start = new Date().getTime();
  if (condition1) {
    return x;
  } else if (condition2) {
    return y;
  }
  var end = new Date().getTime();
  console.log(“took: “ + (end - start));
}
```

So how can we perf test our code?

Combine several well-known Node.JS tools that depend on a common API (Mozilla SpiderMonkey Parser API)

Esprima

Estraverse

Escodegen

Parse the code that we want to test/ analyze/refactor

```
var ast = esprima.parse(“var answer = 42”);
```

```
{
    type: 'Program',
    body: [
        {
            type: 'VariableDeclaration',
            declarations: [
                {
                    type: 'AssignmentExpression',
                    operator: =,
                    left: {
                        type: 'Identifier',
                        name: 'answer'
                    },
                    right: {
                        type: 'Literal',
                        value: 42
                    }
                }
            ]
        }
    ]
}
```

Traverse the AST, compile mini sub-trees for the perf. measurement logic, and inject them in the original tree

```
estraverse.traverse(ast, {
  enter: function (node, parent) {
  },
  leave: function (node, parent) {
    if (node.type === "Program") {
      astToInject = esprima.parse(codeToInject);
      node.body.unshift(astToInject);
    }
  }
});
```

Generate instrumented code

```
var result = escodegen.generate(modifiedAST);
var stream = fs.createWriteStream(fname +
".gen.js");
stream.once("open", function (fd) {
  stream.write(result);
  stream.end();
});
```

Run our app with refs to the instrumented code…

- Store all results in the window (or any other object)
- Have some script serialize results (JSON) and send them to a collection server - REST API
- Visualizer/Reporter - call collection service, get data, visualize it.

No modifications required to the referenced src code

Sending results

```
function _sendPerfData () {
  //serialize and send contents of window._p to
the server which listens for data
  $.post("http://localhost:12346/api/collect", {
    contentType : 'application/json',
    dataType: "json",
    data: JSON.stringify(window._p)
  });
}
```

What can we measure?

- Avg fn time, total exec time, fn count (incremental avg, not standard)
- Call stacks
- Custom stats - how much time is spent in jQuery API vs our own logic

Visualizing results
= separate webapp

1. Query collection server

2. Normalize data - i.e. sort it, take top 10, etc.

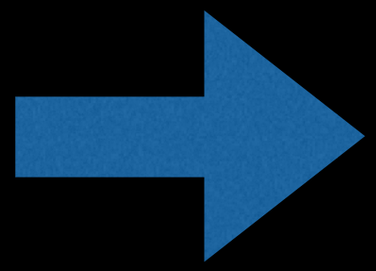
3. Load it into bar charts

4. Load call stack into tree UI (js widget)

Architecture

Instrumentation - CI / Build time

Run tool on src code

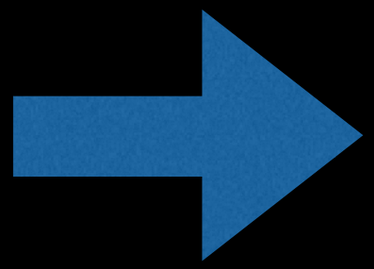


App under test references instrumented code

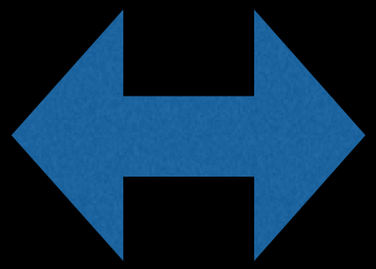
Runtime

Run and **interact with app under test**, send results (client collection script)

JSON



Collection Service
Express.JS
HTTP Server



Query service & visualize results

Some technical aspects

- Need to keep track of return statements location
- If there is no return, we still want to inject at the end of the fn
  - keep a stack of call expressions, push/pop from the stack
  - Keep track of scope/fn declaration in order to resolve correct function names instead of assume anonymous

```
function doWork() {
  if (condition1) {
    return x;
  } else if (condition2) {
    return y;
  }
  var z = 1;
  return z;
}
```

Fn definition examples

- function x () { }
- this.x = function () { }
- var x = function () …

Some technical aspects

- lots of perf statements, avoid repeating ourselves
- use util functions

  : __putstat(‘__renderRecords’, __ms);

Instrumentation overhead at runtime

About the project

Cheetah.JS

- Public on GitHub: https://github.com/attodorov/cheetah.js

- Side project of mine

- grunt-task: https://github.com/attodorov/grunt-cheetah

- Lots of ideas for new functionality


BRACE YOURSELVES
LIVE DEMO IS
NEXT
memegenerator.net

But it doesn't stop here…

- I've also created a cheetah Node.js module that can be used in Mocha/Chai node unit tests
- No requirement for a collection server
- collected data from instrumentation lives in the node process space

CI integration

- grunt task (grunt-cheetah) instruments the code before tests run
- perf tests retrieve exec. times from cheetah global objects
- data is stored & compared with previous runs using **parse.com**
- Travis CI fails the build if the respective perf test fails

Travis CI – Free Hosted Continuous Integration Platform for the Open Source Community

https travis-ci.org/attodorov/blast.js

Reader

travis Home Blog Status Help Travis CI for Private Repositories Angel Todorov

Search all repositories

My Repositories Recent +

attodorov/blast.js 16

uration: 38 sec

Finished: about 21 hours ago

attodorov/cheetah.js 11

uration: 4 sec

Finished: a day ago

attodorov/blast.js

build passing

Blast is a tiny and powerful two-way databinding framework written in JavaScript

Current Build History Pull Requests Branch Summary

master - making sure Travis CI builds only with the latest stable node.JS

#16 passed

ran for 38 sec
about 21 hours ago

Angel authored and committed

Commit d2b50c7 Compare 843dc08..d2b50c7

```
1  Using worker: worker-linux-11-2.bb.travis-ci.org:travis-linux-16
2
3  Build system information                                                    system_info
5
6  $ git clone --depth=50 --branch=master git://github.com/attodorov/blast.js.git attodorov/blast.js   git.checkout   0.64s
16 $ nvm install 0.10.33                                                                                 1.60s
17 ######################################################################## 100.0%
18 Now using node v0.10.33
19 $ node --version
20 v0.10.33
21 $ npm --version
22 1.4.28
23 $ nvm --version
```

Deciding when to fail a performance test

- don’t compare with hardcoded milliseconds, “as is”
- have some threshold / delta
- compare with several previous runs, not just the last one
- normalize stats for different hardware

Handle scopes/
naming better

Add hooks (events) for custom measurements

Use

window.performance

Filters support

Thank You!

Questions?