Sam Stephenson
37signals
Better JavaScript with CoffeeScript
CoffeeScript is Beautiful & I Never Want to Write Plain JavaScript Again
A Brief Personal History
names =
    people.map do |person|
        person.name.capitalize
    end
var names = [];
for (var i = 0, l = people.length; i < l; i++) {
    var person = people[i];
    var name = person.name.slice(0, 1).toUpperCase() + person.name.slice(1);
    names.push(name);
}
prototype
var names = people.map(
    function(person) {
        return person.name.capitalize();
    }
);
var names = people.map(function(person) {
    return person.name.capitalize();
});
var names = people.map(function(person) {
    return person.name.capitalize();
});
We’re Stuck With JavaScript
Compile to JavaScript
Google Web Toolkit
package com.example.gwt.helloworld.client;

import com.google.gwt.core.client EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui Label;
import com.google.gwt.user.client.ui RootPanel;

public class HelloWorld implements EntryPoint {

    @Override
    public void onModuleLoad() {
        Label label = new Label("Hello world");
        Button button = new Button("Say something");
        button.addClickHandler(new ClickHandler() {
            @Override
            public void onClick(ClickEvent event) {
                Window.alert("Hello again");
            }
        });

        RootPanel.get().add(label);
        RootPanel.get().add(button);
    }
}
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='com_example_gwt_helloworld'>
   <!-- Inherit the core Web Toolkit stuff. -->
   <inherits name='com.google.gwt.user.User'/>

   <inherits name='com.google.gwt.user.theme.standard.Standard'/>

   <!-- Specify the app entry point class. -->
   <entry-point class='com.example.gwt.helloworld.client.HelloWorld'/>
</module>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

   <!-- Default page to serve -->
   <welcome-file-list>
      <welcome-file>Com_example_gwt_helloworld.html</welcome-file>
   </welcome-file-list>
</web-app>
Development Mode requires the Google Web Toolkit Developer Plugin

By downloading, you agree to the Terms & Conditions and Privacy Policy.

Download the GWT Developer Plugin For Firefox

plugins for other systems

For help or troubleshooting, ask questions in the discussion group.
Pyjamas
import pyjd # this is dummy in pyjs.
from pyjamas.ui.RootPanel import RootPanel
from pyjamas.ui.Button import Button
from pyjamas.ui.HTML import HTML
from pyjamas.ui.Label import Label
from pyjamas import Window
import pygwt

def greet(fred):
    fred.setText("No, really click me!")
    Window.alert("Hello, AJAX!")

if __name__ == '__main__':
    pyjd.setup("public/Hello.html?fred=foo#me")
    b = Button("Click me", greet, StyleName='teststyle')
    h = HTML("<b>Hello World</b> (html)", StyleName='teststyle')
    l = Label("Hello World (label)", StyleName='teststyle')
    base = HTML("Hello from %s" % pygwt.getModuleBaseURL(),
                StyleName='teststyle')

    RootPanel().add(b)
    RootPanel().add(h)
    RootPanel().add(l)
    RootPanel().add(base)
    pyjd.run()
I love/prefer {insert AJAX / Javascript framework here}, how do I use it?

Not being funny or anything, but unless you have the resources of google or lots of money or lots of time, or you can gather enough people to make it so that everyone has less work to do: you don't.

huh? why?? Some of the widgets in DojoX / Ext-JS are really cute! I want them! waaah!

You are not in Kansas any more. Pyjamas is declarative-style programming, using a "real" programming language. All those widget-sets were designed to be driven from inside HTML (a style of web development which, using Pyjamas, you have just left far behind) and by inserting javascript snippets into the HTML. If you try that with a Pyjamas app, you are not only likely to get yourself into an awful mess, but also you will be unlikely to run the same (python-based) app under Pyjamas-Desktop, as you will still have a Javascript dependency. You can run javascript under Pyjamas-Desktop but it's not easy to interact with (the python code) which is why we went to all the trouble of replacing all the javascript with equivalent python, doing exactly the same things with python-DOM bindings rather than javascript-DOM bindings.

Shoe-horning an alien AJAX toolkit into Pyjamas takes a considerable amount of attention to detail, and is also likely to have unintended side-effects as the two interact. Not only that, but you are in for a bit of a shock when you actually start looking at the number of lines of code in some of these javascript "rich" widget tooklits, and an even bigger one when it comes to linking the two together.
I love/prefer {insert AJAX / Javascript framework here}, how do I use it?

Not being funny or anything, but unless you have the resources of google or lots of money or lots of time, or you can gather enough people to make it so that everyone has less work to do: you don't.

huh? why?? Some of the widgets in DojoX / Ext-JS are really cute! I want them! waaah!

You are not in Kansas any more. Pyjamas is declarative-style programming, using a "real" programming language. All those widget-sets were designed to be driven from inside HTML (a style of web development which, using Pyjamas, you have just left far behind) and by inserting javascript snippets into the HTML. If you try that with a Pyjamas app, you are not only likely to get yourself into an awful mess, but also you will be unlikely to run the same (python-based) app under Pyjamas-Desktop, as you will still have a Javascript dependency. You can run javascript under Pyjamas-Desktop but it's not easy to interact with (the python code) which is why we went to all the trouble of replacing all the javascript with equivalent python, doing exactly the same things with python-DOM bindings rather than javascript-DOM bindings.

Shoe-horning an alien AJAX toolkit into Pyjamas takes a considerable amount of attention to detail, and is also likely to have unintended side-effects as the two interact. Not only that, but you are in for a bit of a shock when you actually start looking at the number of lines of code in some of these javascript "rich" widget tooklits, and an even bigger one when it comes to linking the two together.
I love/prefer {insert AJAX / Javascript framework here}, how do I use it?

Not being funny or anything, but unless you have the resources of google or lots of money or lots of time, or you can gather enough people to make it so that everyone has less work to do: you don't.

huh? why?? Some of the widgets in DojoX / Ext-JS are really cute! I want them! waaah!

You are not in Kansas any more. **Pyjamas is** declarative-style programming, using a "real" programming language. All those widget-sets were designed to be driven from inside HTML (a style of web development which, using Pyjamas, you have just left far behind) and by inserting javascript snippets into the HTML. If you try that with a Pyjamas app, you are not only likely to get yourself into an awful mess, but also you will be unlikely to run the same (python-based) app under Pyjamas-Desktop, as you will still have a Javascript dependency. You can run javascript under Pyjamas-Desktop but it's not easy to interact with (the python code) which is why we went to all the trouble of replacing all the javascript with equivalent python, doing exactly the same things with python-DOM bindings rather than javascript-DOM bindings.

Shoe-horning an alien AJAX toolkit into Pyjamas takes a considerable amount of attention to detail, and is also likely to have unintended side-effects as the two interact. Not only that, but you are in for a bit of a shock when you actually start looking at the number of lines of code in some of these javascript "rich" widget toolkits, and an even bigger one when it comes to linking the two together.
Objective-J
@implementation AppController : CPObj...
CoffeeScript is a little language that compiles into JavaScript. Underneath all of those embarrassing braces and semicolons, JavaScript has always had a gorgeous object model at its heart. CoffeeScript is an attempt to expose the good parts of JavaScript in a simple way.

The golden rule of CoffeeScript is: “It’s just JavaScript”. The code compiles one-to-one into the equivalent JS, and there is no interpretation at runtime. You can use any existing JavaScript library seamlessly (and vice-versa). The compiled output is readable and pretty-printed, passes through JavaScript Lint without warnings, will work in every JavaScript implementation, and tends to run as fast or faster than the equivalent handwritten JavaScript.
CoffeeScript is a little language that compiles into JavaScript. Underneath all of those embarrassing braces and semicolons, JavaScript has always had a gorgeous object model at its heart. CoffeeScript is an attempt to expose the good parts of JavaScript in a simple way.

The golden rule of CoffeeScript is: “It’s just JavaScript”. The code compiles one-to-one into the equivalent JS, and there is no interpretation at runtime. You can use any existing JavaScript library seamlessly (and vice-versa). The compiled output is readable and pretty-printed, passes through JavaScript Lint without warnings, will work in every JavaScript implementation, and tends to run as fast or faster than the equivalent handwritten JavaScript.
It’s Just JavaScript

$(function() {
    $('body').html("Hello world");
});
It’s Just JavaScript

$ ->

$ ("body").html "Hello world"
CoffeeScript is a little language that compiles into JavaScript. Underneath all of those embarrassing braces and semicolons, JavaScript has always had a gorgeous object model at its heart. **CoffeeScript is an attempt to expose the good parts of JavaScript in a simple way.**

The golden rule of CoffeeScript is: “It’s just JavaScript”. The code compiles one-to-one into the equivalent JS, and there is no interpretation at runtime. You can use any existing JavaScript library seamlessly (and vice-versa). The compiled output is readable and pretty-printed, passes through JavaScript Lint without warnings, will work in every JavaScript implementation, and tends to run as fast or faster than the equivalent handwritten JavaScript.
Presenters and thought leaders: the "compare the size of JavaScript: The Definitive Guide and JavaScript: The Good Parts" joke isn't funny
Good Part: Private by default
Good Part:
Private by default

(function() {
    /* your program here */
}).call(this);
Good Part:
No more var
Good Part:
No more var

```
lastClick = 0
$("a").click ->
    now = new Date().getTime()
    if now - lastClick > 100
        $('#message').show()
    lastClick = now
```
Good Part:
No more `var`

```javascript
var lastClick = 0
$('a').click ->
    var now = new Date().getTime()
    if now - lastClick > 100
        $('#message').show()
    lastClick = now
```
Good Part:
Strict comparisons
Good Part:
Strict comparisons

== VS. ===
Good Part:
Strict comparisons

"true" == true  // true
"true" === true  // false
"0" == false    // true
"0" == 0       // true
0 === false    // false
"" == false    // true
Good Part:
Strict comparisons

"true" is true        // false
"true" is "true"      // true
"0" is false         // false
"0" is 0             // false
0 is 0               // true
"" is false          // false
Good Part:
Runs anywhere
Good Part:
Runs anywhere
JSLint compliant
10 Things I Love About CoffeeScript
1. Function Syntax
1. Function Syntax

```javascript
function greet (name) {
    return "Hello " + name;
}
```
1. Function Syntax

greet = (name) ->
  "Hello " + name
1. Function Syntax

```javascript
$("a").click(function(event) {
    $(this).addClass("busy");
});
```
1. Function Syntax

$("a").click (event) ->
    $(this).addClass "busy"
2. Significant Whitespace
if (url) {
    $.get(url, function(data) {
        return $('#result').html(data);
    });
} else {
    $('#error').show();
}
2. Significant Whitespace

```javascript
if url
  $.get url, (data) ->
    $("#result").html data
else
  $("#error").show()
```
2. Significant Whitespace

```javascript
var readConfiguration = function(callback) {
    return path.exists(filename, function(err, exists) {
        if (exists) {
            return fs.readFile(filename, callback);
        } else {
            return callback(false);
        }
    });
};
```
2. Significant Whitespace

```javascript
readConfiguration = (callback) => {
    path.exists filename, (err, exists) => {
        if (exists)
            fs.readFile filename, callback
        else
            callback false
    }
}
```
3. Bare Objects
3. Bare Objects

$(this).css({ top: "20px", left: "-20px" });
3. Bare Objects

\`\$(this).css top: "20px", left: "-20px"\`
3. Bare Objects

$.ajax({
    url: path,
    timeout: 5,
    data: {
        from: "workspace"
    },
    dataType: "html",
    success: function(data) {
        return $('#result').html(data);
    }
});
3. Bare Objects

$.ajax
    url: path,
    timeout: 5,
    data:
        from: "workspace",
    dataType: "html",
    success: (data) ->
        $("#result").html data
3. Bare Objects

```python
person =
    name: "Sam"
    age: 27
    profession: "Programmer"
```
4. Everything’s an Expression
4. Everything’s an Expression

switch keyCode
    when 38
        command = "previous"
    when 40
        command = "next"
    when 13
        command = "select"
4. Everything’s an Expression

\[
\text{command} = \text{switch keyCode}
\begin{align*}
&\text{when 38 then } \text{"previous"} \\
&\text{when 40 then } \text{"next"} \\
&\text{when 13 then } \text{"select"}
\end{align*}
\]
getCommand = (keyCode) ->
    switch keyCode
    when 38 then "previous"
    when 40 then "next"
    when 13 then "select"
5. Comprehensions
5. Comprehensions

```python
names = []
for person in people
    names.push capitalize person.name
```
5. Comprehensions

```python
capitalize person.name
```
5. Comprehensons

ages = (person.age for person in people)
5. Comprehensions

names = for person in people when age > 27
capitalize person.name
var names, person;
names = (function() {
    var _i, _len, _results;
    _results = [];
    for (_i = 0, _len = people.length; _i < _len; _i++) {
        person = people[_i];
        if (age > 27) {
            _results.push(capitalize(person.name));
        }
    }
    return _results;
})()
6. Classes & Inheritance
6. Classes & Inheritance

class Photo
constructor: (url) ->
  this.url = url

createElement: ->
  $("<img>").attr "src", this.url
6. Classes & Inheritance

class Photo
    constructor: (url) ->
        this.url = url

    createElement: ->
        $("<img>").attr "src", this.url
6. Classes & Inheritance

class Photo
    constructor: (url) ->
        @url = url

    createElement: ->
        $("<img>").attr "src", @url
6. Classes & Inheritance

class Photo
    constructor: (@url) ->

    createElement: ->
        $"<img>".attr "src", @url
6. Classes & Inheritance

class Thumbnail extends Photo
    createElement: ->
        $el = super
        $el.height 100
var Thumbnail;
var __hasProp = Object.prototype.hasOwnProperty, __extends = function(child, parent) {
    for (var key in parent) { if (__hasProp.call(parent, key)) child[key] = parent[key]; }
    function ctor() { this.constructor = child; }
    ctor.prototype = parent.prototype;
    child.prototype = new ctor;
    child.__super__ = parent.prototype;
    return child;
};

Thumbnail = (function() {
    __extends(Thumbnail, Photo);
    function Thumbnail() {
        Thumbnail.__super__.constructor.apply(this, arguments);
    }
    Thumbnail.prototype.createElement = function() {
        var $el;
        $el = Thumbnail.__super__.createElement.apply(this, arguments);
        return $el.height(100);
    }
    return Thumbnail;
})(());
7. Bound Functions
class PersonView
    constructor: (@person, @el) ->

    request: ->
        $.get @person.url, (data) ->
            $(@el).html data
7. Bound Functions

class PersonView

  constructor: (@person, @el) ->

  request: ->
    $.get @person.url, (data) ->
      $(@el).html data
function request () {
    $.get(this.person.url, (function(data) {
        $(this.el).html(data);
    })).bind(this)
}
7. Bound Functions

class PersonView
    constructor: (@person, @el) ->

    request: ->
        $.get @person.url, (data) =>
            $(@el).html data
7. Bound Functions

class PersonView

constructor: (@person, @el) ->

$(@el).bind "click", @showName

showName: =>

  $(@el).html @person.name
8. Conditionals
8. Conditionals

@request() if @isActive()

return unless $("li").length
8. Conditionals

result + 10 if result?
8. Conditionals

@panel?.restore()
8. Conditionals

@panel.url !== window.location
9. Destructuring Assignment
9. Destructuring Assignment

```javascript
name = person.name
```
9. Destructuring Assignment

```javascript
name = person.name
{name} = person
```
9. Destructuring Assignment

name = person.name
{name} = person
{name, age} = person
9. Destructuring Assignment

[first, last] = person.name.split " "
10. String Syntax
10. String Syntax

greet = (name) ->
    "Hello #{name}"
10. String Syntax

greet = (name) ->
  "Hello #{name.toUpperCase()}"
10. String Syntax

```javascript
render = (person) =>
    ""
    <div class="person">
        <a href="#{person.url}">#{person.name}</a>
        <span>#{person.profession}</span>
    </div>
    ""
```
How To
Use It
CoffeeScript is a little language that compiles into JavaScript. Underneath all of those embarrassing braces and semicolons, JavaScript has always had a gorgeous object model at its heart. CoffeeScript is an attempt to expose the good parts of JavaScript in a simple way.

The golden rule of CoffeeScript is: *it’s just JavaScript*. The code compiles one-to-one into the equivalent JS, and there is no interpretation at runtime. You can use any existing JavaScript library seamlessly (and vice-versa). The compiled output is readable and pretty-printed, passes through JavaScriptLint without warnings, will work in every JavaScript implementation, and tends to run as fast or faster than the equivalent handwritten JavaScript.

Latest Version: 1.1.2

Overview

CoffeeScript on the left, compiled JavaScript output on the right.

```coffee
# Assignment:
number = 42
opposite = true

# Conditions:
number = 42 if opposite

# Functions:
square = (x) -> x * x
```

```javascript
var cubes, list, math, num, number, opposite, race, square;
var __slice = Array.prototype.slice;
number = 42;
opposite = true;
if (opposite) number = -42;
square = function(x) {
  return x * x;
};
list = [1, 2, 3, 4, 5];
```
Command Line

$ npm install -g coffee-script
$ coffee -c hello.coffee

$ cat hello.js
(function() {
  alert("hello");
}).call(this);
Command Line

$ coffee -cw hello.coffee
Rails 3.1
Sprockets

https://github.com/sstephenson/sprockets
Node.js
Stitch

https://github.com/sstephenson/stitch
Thank You

@sstephenson
CoffeeScript

http://coffeescript.org/