amadeus

create your DSL with PEG.js !

Sophia Conf 2014

Bertrand Laporte
Sophia-Antipolis
July 3rd, 2014

Domain-specific language – *aka. DSL (noun):*

A computer programming language of limited expressiveness focused on a particular domain.

martin fowler

Examples:

HTML CSS LESS RegExp SVG

VHDL YACC XML YAML JSON

amadeus

why do we use DSLs?

productivity

less code to write, more simple paradigm
ex: HTML, Markdown, YAML

quality

less errors / clear error reporting, validation tools
ex: XML, HTML

specialization

higher level language / no need to know the underlying technology, lower learning curve, expand to different skill sets
ex: Rules, XML, SVG, VHDL , Excel macros

custom DSLs – what for?

rule engines

- simple macros (spreadsheet)
- customization rules written by business analysts

test & validation

- high-level test scripts written by functional specialists
- configuration validation

data-extraction

- screen scrapping
- multiple data format support

how to create a DSL?

- method #1:
  rely on another language
  (pseudo-DSLs)

```
describe('Array', function(){

  describe('#indexOf()', function(){
    it('should return -1 when the value is not present', function(){
      expect([1,2,3].indexOf(5)).to.equal(-1);
      expect([1,2,3].indexOf(0)).to.equal(-1);
    })
  });

});
```

how to create a DSL?

- method #1:
  rely on another language
  (pseudo-DSLs)
- method #2:
  grammar + parser + code generator
  (full DSLs)

⇩

PEG.js

```
{template hello(name)}
    <input value="{name}" placeholder="enter your name">
    <div class="msg">
        Hello {name}!
        {if name == "world"}
            <strong> Welcome to hashspace </strong>
        {/if}
    </div>
{/template}

// display the template in the #output div
hello("").render("output");
```


amadeus

PEG.js

Parser Generator for JavaScript

http://pegjs.majda.cz

PEG: Parsing Expression Grammar

- an alternative to Context Free Grammars and Regular Expressions for describing machine-oriented syntax
- introduced by **Bryan Ford** (Yale) in **2004** and is closely related to the family of top-down parsing languages introduced in the early 1970s

PEG.js: PEG parser generator for Javascript

- Developed & maintained by **David Majda** since March 2010

PEG.js
in action

Sample:
Simple JSON parser
(only object and string values)

```
"123"
{"foo":"234","bar":{"foo2":"Hello Sophia!"}}
```

Key concepts:
rules, labels, regexp & predicates

```
// simple_json.pegjs - code derived from D. Majda JSON grammar
JSON_value
  = ws v:value ws { return v; }

ws "whitespace"
  = [ \t\n\r]*

value
  = object / string

object
  = ws "{" ws
    members:(
      first:member
      rest:(ws "," ws m:member { return m; })*
      {
        var result = {}, i;
        result[first.name] = first.value;
        for (i = 0; i < rest.length; i++) {
          result[rest[i].name] = rest[i].value;
        }
        return result;
      }
    )?
    ws "}" ws
    { return members !== null ? members: {}; }

member
  = nm:string ws ":" ws v:value { return { name: nm, value: v }; }

string "string"
  = '"' chars:character* '"' { return chars.join(""); }

character
  = [\x20-\x21\x23-\x5B\x5D-\u10FFFF]

/* TODO support number, boolean, array and string escape sequences */
```

PEG.js
in action

#1 generate parser
#2 use it in sample script
#3 run sample script

simple_json.peg.js:
660 lines / 18k

```
$ grunt peg
Running "peg:grammar" (peg) task
Generating parser from "src/simple_json.pegjs"...
Parser "src/simple_json.peg.js" generated in 19ms.

Done, without errors.
blaporte@NCEL498 /d/Home/peg-pres/src
```

```
// sample.js
var parser = require("./simple_json.peg.js");

var samples=[
    '"123"',
    '{"foo":"234","bar":{"foo2":"Hello Sophia!"}}'
];

for (var i=0;samples.length>i;i++) {
    console.log( (i+1) + "." + JSON.stringify( parser.parse(samples[i]) ) );
}
```

```
$ node sample
1."123"
2.{"foo":"234","bar":{"foo2":"Hello Sophia!"}}
blaporte@NCEL498 /d/Home/peg-pres/src
$
```


amadeus

real life example

PEG.js

Parser Generator for JavaScript

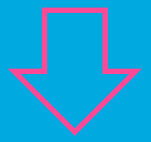
for

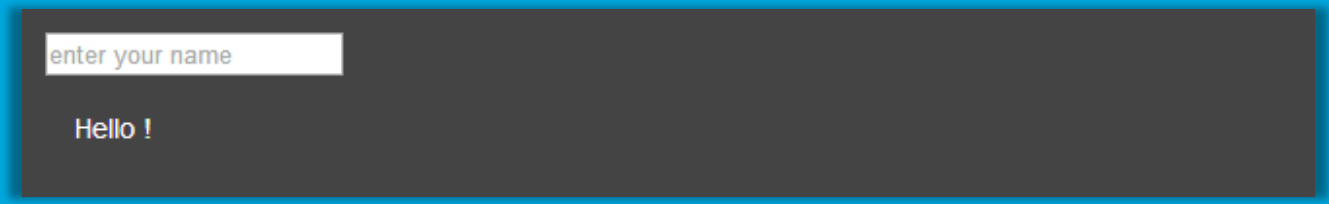
#hashspace

#hashspace
overview

- simple syntax

```
{template hello(name)}
    <input value="{name}" placeholder="enter your name">
    <div class="msg">
        Hello {name}!
        {if name == "world"}
            <strong> Welcome to hashspace </strong>
        {/if}
    </div>
{/template}

// display the template in the #output div
hello("").render("output");
```



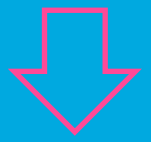


amadeus

#hashspace
overview

- simple syntax
- data-binding

```
{template hello(name)}
    <input value="{name}" placeholder="enter your name">
    <div class="msg">
        Hello {name}!
        {if name == "world"}
            <strong> Welcome to hashspace </strong>
        {/if}
    </div>
{/template}

// display the template in the #output div
hello("").render("output");
```



world

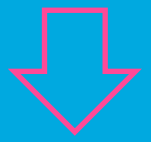
Hello world! **Welcome to hashspace**

amadeus

#hashspace
overview

- simple syntax
- data-binding
- error-handling

```
{template hello(name)}
    <input value="{name}" placeholder="enter your name">
    <div class="msg">
        Hello {name}!
        {if name == "world"} <
            <strong> Welcome to hashspace </strong>
        {/if}
    </div>
{/template}

// display the template in the #output div
hello("").render("output");
```



- Invalid HTML element syntax

  [hello.hsp] line: 5, column: 30

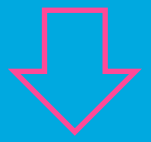
  >> code: <

amadeus

#hashspace
overview

- simple syntax
- data-binding
- error-handling
- easy widgets/components

```
var panel=require("./panel.hsp");

{template test(m)}
  <#panel body="Panel A (headless): {m.text}"/>

  <#panel head="Panel B ({m.text}!)">
    {m.text}! <a onclick="{update(1)}">Update model</a>
  </#panel>

  <#panel>
    <@head>Panel C: <a onclick="{update(10)}">Update model</a></@head>
    <@body>{m.text}! <a onclick="{update(100)}">Update model</a></@body>
  </#panel>
{/template}
```



Panel A (headless): Hello World

Panel B (Hello World!)
Hello World! Update model

Panel C: Update model
Hello World! Update model

amadeus

#hashspace
overview

- simple syntax
- data-binding
- error-handling
- easy widgets/components

```
{template test(d)}
  Click on an item to select it:

  <#list head="Static list" class="listcpt" onselect="{showSelection(event.value)}">
    <@option value="A" label="First {d.itemName}"/>
    <@option value="B">Second {d.itemName}</@option>
  </#list>

  <#list class="listcpt" onselect="{showSelection(event.value)}">
    <@head>
      Dynamic list:
      <a onclick="{empty()}">Empty</a> -
      <a onclick="{update()}">Update list</a>
    </@head>
    {foreach idx,itm in d.items}
      <@option value="K{idx}">{idx+1}. {itm}</@option>
    {/foreach}
  </#list>

  {if d.selectedItem!==null}Last selected value: {d.selectedItem}{/if}
{/template}
```

Click on an item to select it:

Static list

- First item
- Second item

Dynamic list: Empty - Update list

- 1. Kind of blue
- 2. Something else
- 3. Winter moon

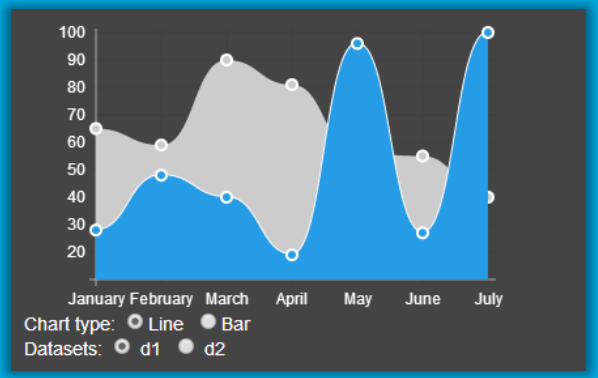
Last selected value: K0

amadeus

#hashspace

overview

- simple syntax
- data-binding
- error-handling
- easy widgets/components
- easy 3rd party integration



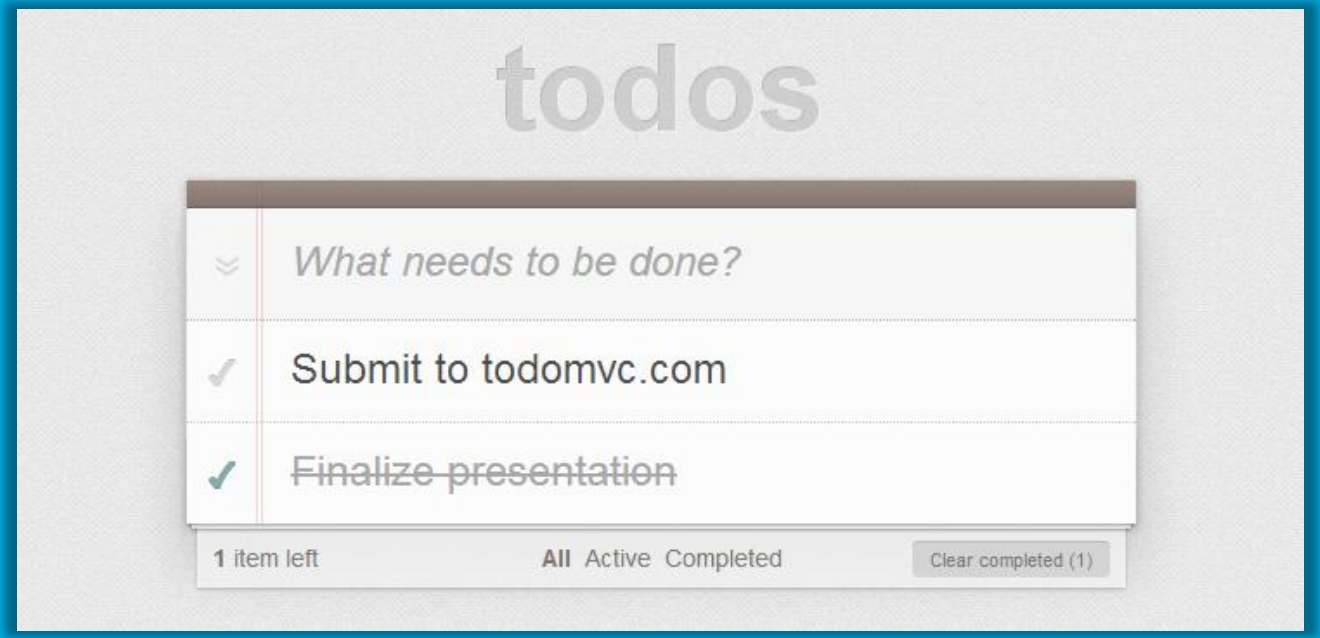

```
var ChartCpt=klass({
  attributes:{
    width:{type:"int",binding:"1-way",defaultValue:100},
    height:{type:"int",binding:"1-way",defaultValue:100},
    type:{type:"string",binding:"1-way",defaultValue:"line"},
    labels:{type:"object",binding:"1-way",defaultValue:[]},
    datasets:{type:"object",binding:"1-way",defaultValue:[]},
    options:{type:"object",defaultValue:{scaleFontColor:"#fff",scaleLineColor:"#AAA"}}
  },
  $refresh:function() {
    if (!this.chart) {
      var canvas=this.$getElement(0);
      this.chart=new Chart(canvas.getContext("2d"));
    }
    if (this.type==="bar") {
      this.chart.Bar({labels:this.labels,datasets:this.datasets},this.options);
    } else if (this.type==="line") {
      this.chart.Line({labels:this.labels,datasets:this.datasets},this.options);
    } else {
      log.error("[ChartCpt] Invalid type: "+this.type);
    }
  }
});

{template chart using c:ChartCpt}
  <canvas width="{c.width}" height="{c.height}"></canvas>
{/template}
```

```
{template test(data)}
  {let datasets=(data.ds=="d1")? d1 : d2}
  <#chart width="380" height="220" type="{data.type}" labels="{labels}" datasets="{datasets}"/>
  <div style="padding:15 0 0 30">
    // Chart type selection here
  </div>
{/template}
```


amadeus

#hashspace
overview

- simple syntax
- data-binding
- error-handling
- easy widgets/components
- easy 3rd party integration
- compact !



http://hashspace.ariatemplates.com/todomvc

Framework	Fwk. size	App. size
jQuery + handlebar	32kb (27+5)	~ 3.8kb
hashspace + noderJS	25kb (19+6)	~ 2.2kb
no framework (vanilla)	0kb	~10kb

amadeus

PEG.js
key features

- full validation *(grammar)*
- simple syntax
- extensible w/ JavaScript *(pre- / post- processing)*
- run in nodeJS & in browser
- precise errors *(line & column numbers)*
- fast *(with cache=true)*
- JavaScript grammar available *(ES5.1)*

amadeus

PEG.js
some tips...

#1 catch errors in your grammar

```
TemplateContent "template content"
  = _ blocks:(  TplTextBlock
                / CommentBlock / HTMLCommentBlock
                / IfBlock / ElseIfBlock / ElseBlock / EndIfBlock
                / ForeachBlock / EndForeachBlock
                / HTMLElement / EndHTMLElement
                / HspComponent / EndHspComponent
                / HspCptAttribute / EndHspCptAttribute
                / LetBlock
                / LogBlock
                / ExpressionBlock
                / InvalidHTMLElement
                / InvalidBlock)*
  {return blocks}
```

```
InvalidHTMLElement
  = "<" code:[^\r\n]* EOL
  {return {type:"invalidelement", code:'<'+code.join(''), line:line, column:column}}
```

```
InvalidBlock
  = "{" !(_ "/template" _ "}") chars:[^{}#]* "}"
  {return {type:"invalidblock", code:chars.join(''), line:line, column:column}}
```


amadeus

PEG.js
some tips…

#2 use JavaScript post-processing for advanced validation

```
TemplateContent "template content"
  = _ blocks:(  TplTextBlock
                / CommentBlock / HTMLCommentBlock
                / IfBlock / ElseIfBlock / ElseBlock / EndIfBlock
                / ForeachBlock / EndForeachBlock
                / HTMLElement / EndHTMLElement
                / HspComponent / EndHspComponent
                / HspCptAttribute / EndHspCptAttribute
                / LetBlock
                / LogBlock
                / ExpressionBlock
                / InvalidHTMLElement
                / InvalidBlock)*
  {return blocks}
```


amadeus

PEG.js
some tips…

#3 use test-driven development *(aka TDD)* !

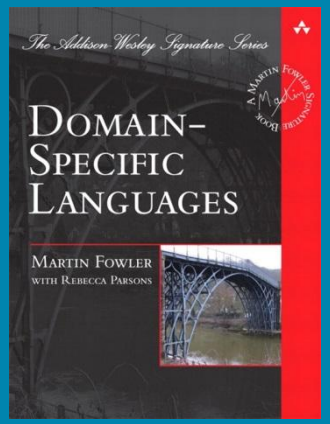
```
##### Template:
{template hello(person,property)}
  {person[person.name].foo}
{/template}

##### Parsed Tree:
"skip"

##### Syntax Tree:
"skip"

##### Template Code
hello=[
  n.$text({
    e1:[7,3,function(i,a0,a1) {return [a0,a1,"foo"][i];},2,3],
    e2:[1,1,"person"],
    e3:[1,2,"person","name"]
  },["",1])
]
```


amadeus

#conclusion

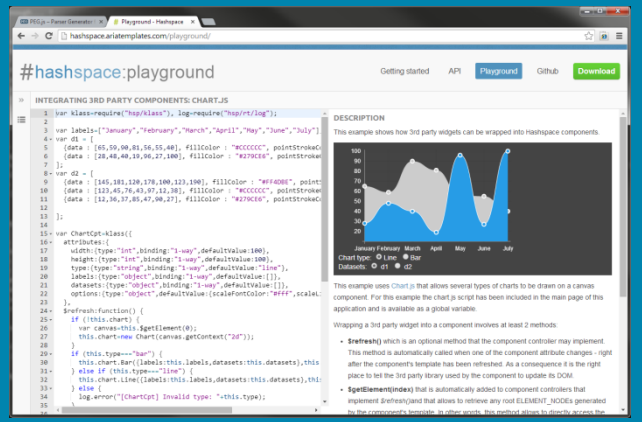
PEG.js rocks!

#more reading



http://martinfowler.com/dsl.html

http://pegjs.majda.cz
http://bford.info/packrat



http://hashspace.ariatemplates.com

amadeus

thank you !

You can follow us on:
AmadeusITGroup

amadeus.com/blog
amadeus.com

amadeus