TEMPLATING

PAST, PRESENT AND FUTURE

Presented by Ed Singleton

singletoned@gmail.com

@Singletoned

WHAT IS TEMPLATING?

A system for generating different strings in different contexts

Context mostly referring to a collection of variables

```
{"foo": 3, "bar": 5}
```

WHAT TYPES OF TEMPLATING ARE THERE?

LOTS!

STRING CONCATENATION

```
'''<div class="''' + foo_class + '''">Hullo</div>'''
```

- Difficult to read and maintain
- Unsafe to run from untrusted sources (obviously!)
- Complicated to correctly escape

STRING CONCATENATION

WHERE'S IT USED?

JavaScript

```
$("<div class=\""+foo_class+"\">hello world</div>")
```

```
# Add 2 links at the bottom for "Previous month", "Next month"
if form.action == 'view':
    prevUrl = COUtil.addToUrl(COUtil.addToUrl(cherrypy.url(cherrypy.request.path_info, cherrypy.request.
        query_string).replace('&', '&amp;'), self.name + '_startMonth', prevMonth), self.name + '_startYear',
        prevYear)
    nextUrl = COUtil.addToUrl(COUtil.addToUrl(cherrypy.url(cherrypy.request.path_info, cherrypy.request.
        query_string).replace('&', '&amp;'), self.name + '_startMonth', nextMonth), self.name + '_startYear',
        nextYear)
    res += """<a href="%s" class="btn"><i class="icon-chevron-left"></i> Prev</a>""" % prevUrl
    res += "</td><td colspan=2> </td><td class='pull-right'>"
    res += """<a href="%s">Next <i class="icon-chevron-right"></i></a>""" % nextUrl
else:
    res += """<a href="#" onClick="previousMonths('%s');" class="btn"><i class="icon-chevron-left"></i> Prev</a>
        """ % self.name
    res += "</td><td colspan=2> </td><td class='pull-right'>"
    res += """<a href="#" onClick="nextMonths('%s');" class="btn">Next <i class="icon-chevron-right"></i></a>""" %
         self.name

    res += """<input type=hidden name=%s_weekFirstDay value=''>""" % self.name
    res += """<input type=hidden name=%s_isWeekOn value=''>""" % self.name
    res += """<input type=hidden name=%s_startDragId value=''><input type=hidden name=%s_endDragId value=''>""" %
        (self.name, self.name)
    res += """<input type=hidden name=%s_startMonth value=%s><input type=hidden name=%s_startYear value=%s>""" % (
            self.name, startMonth, self.name, startYear)
```

STRING TEMPLATES

```
"Hullo. Is it %s you're looking for" % foo
"Who let the {things} out?".format(things="Cats")
```

- Structured and Safer
- Logic free

THAT WAS THE PAST

(HOPEFULLY)

EXECUTABLE

- Ruby
  - ERB
- PSP (Python Server Pages)
  - mod_python
  - Spyce
  - Webware
- PHP
- ASP
- JSP
- Python
  - Cheetah
  - Mako

EXECUTABLE

- Allow arbitary code execution:
  - Turing Complete
- Can be used for anything, not just html
- Bracket / Expression / Bracket
- Not safe to run templates from untrusted sources
- The vast majority of templating

INTERPRETED (SANDBOX ENVIRONMENTS)

- PHP Templates
  - Twig
  - Smarty
- Python
  - Django Templates
  - Jinja2
- Ruby
  - Liquid

```
{% extends 'master.html' %}

{% block title %}{{_('View Your Basket')}}{% endblock title %}

{% block mainbody %}
      <div role="main" id="main">
        <div class="basket gradient-diag">
          <h2>{{_('Basket')}}</h2>
          <p>{{utils.event_header(event)}}</p>
          <table class="table" summary="This table outlines the items in the shopping basket.">
            <tbody>
              {% for item in basket['items'] %}
              <tr class="basket-{{item['id']}}">
                <td class="quantity">{{item['quantity']}}</td>
                <td class="wide name">{{item['name']}}</td>
                <td class="price text-right">{{item['price']}}</td>
              </tr>
              {% endfor %}
            </tbody>
            <tfoot class="bold text-upper">
              <tr class="basket-total">
                <td colspan="2">Totalt</td>
                <td class="text-right">{{basket['total']}}</td>
              </tr>
            </tfoot>
          </table>
        </div>
      </div>
      <div class="navigation">
        {{utils.restart_button()}}
        {{utils.prev_button()}}
        <button class="btn btn-primary btn-md width-250 pull-right prev"><img src="images/pictures/credit-card.png
        " alt="" /> Betal</button>
      </div>
{% endblock mainbody %}
```

INTERPRETED (SANDBOX ENVIRONMENTS)

- Safer, no arbitary code execution
- Can run untrusted templates
- Usually a limited syntax
- Don't have full expressive power of the language
- Can still be used for anything, not just html
- Bracket / Expression / Bracket

LOGIC-FREE

- JavaScript
  - Mustache / Handlebars
- Only a slight advance on string templates
  - Adds in loops and conditionals
- Prevents business logic from being captured in templates
  - (And display logic)
- Tends to lead to fatter controllers
- Could be achieved in any other system with enough discipline
- Inherently safe

STRUCTURED (SGML)

- Python
  - ZPT (20 years old)
  - Kid
  - Genshi
- Ruby
  - Radius
  - Amrita
- Java
  - Thymeleaf

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:py="http://genshi.edgewall.org/">

  <title>Geddit: ${link.title}</title>
  <id href="${url('/info/%s/' % link.id)}"/>
  <link rel="alternate" href="${url('/info/%s/' % link.id)}" type="text/html"/>
  <link rel="self" href="${url('/feed/%s/' % link.id)}" type="application/atom+xml"/>
  <updated py:with="time=link.comments and link.comments[-1].time or link.time">
    ${time.isoformat()}
  </updated>

  <?python from genshi import HTML ?>
  <entry py:for="idx, comment in enumerate(reversed(link.comments))">
    <title>Comment ${len(link.comments) - idx} on “${link.title}”</title>
    <link rel="alternate" href="${url('/info/%s/' % link.id)}#comment${idx}"
          type="text/html"/>
    <id>${url('/info/%s/' % link.id)}#comment${idx}</id>
    <author>
      <name>${comment.username}</name>
    </author>
    <updated>${comment.time.isoformat()}</updated>
    <content type="xhtml"><div xmlns="http://www.w3.org/1999/xhtml">
      ${HTML(comment.content)}
    </div></content>
  </entry>

</feed>
```

STRUCTURED (SGML)

- Can't produce strings which are invalid SGML
- Not Turing Complete
- Safer
- Normally safe to run other peoples templates

STRUCTURED (WHITESPACE SIGNIFICANT)

- JavaScript
  - Jade
- Python
  - PyJade
- Ruby
  - HAML
  - Slim

```
doctype html
html
  head
    title Slim Examples
    meta name="keywords" content="template language"
    meta name="author" content=author
    javascript:
      alert('Slim supports embedded javascript!')

  body
    h1 Markup examples

    #content
      p This example shows you how a basic Slim file looks like.

      == yield

      - unless items.empty?
        table
          - items.each do |item|
            tr
              td.name = item.name
              td.price = item.price
      - else
```

```
      p
        | No items found.  Please add some inventory.
          Thank you!

  div id="footer"
    = render 'footer'
    | Copyright © #{year} #{author}
```

STRUCTURED (WHITESPACE SIGNIFICANT)

- Potentially suited to Pythonistas
  - We already tolerate whitespace significance
- Can't produce strings which are invalid SGML
- Not Turing Complete
- Not inherently safe

STRUCTURED (INTERNAL DSL)

- Python
  - Nevow Stan
  - Breve
  - werkzeug.utils.HTMLBuilder
- Can't produce strings which are invalid SGML
- Never quite caught on

```python
from werkzeug.utils import HTMLBuilder

html = werkzeug.utils.HTMLBuilder('html')
result = html.p(
    class_='foo',
    *[
        html.a('foo', href='foo.html'),
        ' ',
        html.a('bar', href='bar.html')
    ]
)

expected = '''
<p class="foo"><a href="foo.html">foo</a> <a href="bar.html">bar</a></p>
'''.strip()

assert result == expected
```

STRUCTURED (BUILDERS)

- Python
  - Elementree Builder
  - lxml.html.builder
- Ruby
  - Hpricot (Deprecated)
  - NokoGiri
- JavaScript
  - DOMBuilder
  - 100s of others
- Racket / Scheme
  - X-Expressions

```python
import lxml.html
from lxml.html.builder import *

html = HTML(
    HEAD(TITLE("Hello World")),
    BODY(CLASS("main"),
        H1("Hello World !")))

html.find('body').append(P("Hullo Moon"))

print lxml.html.tostring(html, pretty_print=True)
```

```
<html>
<head><title>Hello World</title></head>
<body class="main">
<h1>Hello World !</h1>
<p>Hullo Moon</p>
</body>
</html>
```

STRUCTURED (BUILDERS)

- Build internal structures
  - Build, Manipulate, Serialize
  - Allows passing fragments around
- Less readable in terms of document structure

STRUCTURED (NOT SGML):

- Python
  - SQLAlchemy
  - Django ORM
- JavaScript
  - SASS/Less
- Most query languages

A FAILED EXPERIMENT

I tried to do something new, but it didn't really pan out

```
from wiseguy.template import Template, Transform, add, jade
from helpers import (
    add_seat_rows, nav_seat_form, mark_selected_seats, error_message)

template = Template(
    element=jade('''
div#main-body.mini-layout-body
  div#seats
    div#seat-preview-wrapper
      div#seat-preview
        div.screen
        table.seats
    div#seat-chooser-wrapper
      div#seat-chooser
        div(style="display: block;")
          div.screen= "SCREEN"
          table.seats '''),
    transforms=[
        Transform(
            set(["seat_rows"]),
            add_seat_rows),
        Transform(
            set(["tickets"]),
            lambda template, tickets: template.element.set_attr(
                'div#seats',
```

```
                'data-tickets-number',
                str(sum(tickets.values())))),
        Transform(
            set(["url", "seat_ids"]),
            add("ul.nav", nav_seat_form)),
        Transform(
            set(["errors"]),
            lambda template, errors: errors and template.element.add(
                "div#seats",
                error_message(errors["seat_ids"]),
                index=0)),
        Transform(
            set(["seat_ids"]),
            mark_selected_seats)])

expected_keys = set(
    ["seat_rows", "tickets", "url", "seat_ids", "errors"])

assert template.keys() == expected_keys
```

A FAILED EXPERIMENT

- Introspectable
- Highly Performant
- Difficult to read and maintain

WHAT DOES THE FUTURE HOLD?

I DON'T KNOW

THE FUTURE - MY GUESSES

- More DSLs
- Markdown and Jade make a wonderful pair
- SGML is well served, but what about others?

THE FUTURE - MY GUESSES

- Generating whitespace significant strings with a whitespace ignorant template system is painful
- Jinja2 is not the best choice for Ansible and Salt
- Another DSL for DevOps is likely
  - (Puppet already has one)

THE FUTURE - MY GUESSES

- GraphQL is kind of a templating language
- JSON templates to map database rows into JSON objects for APIs?

THE FUTURE - MY GUESSES

- Anything verbose is a target for templating
  - Template Java code?

THE END

ANY QUESTIONS?