Nevow: Web App Construction Kit

Donovan Preston
Divmod

Why Nevow?

- HTML Templating
- Built on twisted.web
- No code in HTML
- Rich Python DOM manipulation API
  - Not the W3C DOM :-)

History – Woven

- HTML Templates with TAL-like attributes
  - “directives”
- W3C DOM for transforming the template
  - Like XMLC (Java)
- Abstract Widget and Controller layers
- Complex

DIVMOD

Nevow

- Much lighter weight
- Simple Python DOM called "stan"
- Lets you use the full power of Python to write your view logic
- Faster, easier to use, shorter code

Demo: Sched

Nevow Concepts

- Object Publishing
  - Maps every URL to a Page instance
- Template Driven
  - Pages generate HTML from a template
- Directives and Patterns
- Uses a context stack to keep render state

DIVMOD

Object Publishing

- The URL /foo/bar/baz is split into segments
  - (’foo’, ‘bar’, ‘baz’)
- Page.locateChild(request, segments) called
  - Return value is (Page, segments)
- When no segments left:
  - Page.render(request)

Directives

- Page.render(request) loads docFactory
  - Template document is processed, resulting in final output
- Template can invoke Python code:
  - <span nevow:data=”foo”> → def data_foo(self, ctx, data):
  - <span nevow:render=”bar”> → def render_bar(self, ctx, data):

Patterns and Slots

- Inside a Python render method:
  - Copy a template fragment “pattern”
    - ctx.onePattern(’baz’) → <span nevow:pattern=”baz”>
  - Fill a placeholder “slot”
    - ctx.fillSlot(’qux’, newValue) → <nevow:slot name=”qux”>

Context Stack

- A Context instance is created for every dynamic node which is rendered
  - Context nodes are chained up to the top
- ctx.tag is the current template node
- ctx.remember(interface, implementation)
- ctx.locate(interface)

Example 1

Tutorial: Sched

- Start by adding directives to HTML

```
<span nevow:data="currentMonth" nevow:render="month">
    <h1><nevow:slot name="label">The label goes here</nevow:slot></h1>
    <table height="50%" width="50%" border="1">
        <tr>
            <td>Sunday</td>
            ...
            <td>Saturday</td>
        </tr>
        <tr nevow:pattern="calendarWeek" nevow:render="remove">
            <td nevow:pattern="calendarDay" align="center"></td>
            ...
        </tr>
            <nevow:slot name="calendarBody" />
    ...
```

Example 1

Data and Render

- A Page class is responsible for rendering one URL (one web page)
- **data** and **render** methods are invoked by **directives**

```
<span nevow:data="currentMonth" nevow:render="month">
                      ↓
class ScheduleRoot(rend.Page):
    docFactory = rend.htmlfile('Month.html')

    def data_currentMonth(self, context, data): ...

    def render_month(self, context, data): ...
```

Data Methods

- Data method retrieve or produce data
- The result is passed as “data” while this node and it’s children are rendered

```
def data_currentMonth(self, context, data):
    curtime = time.localtime()
    year = int(context.arg('year', curtime[0]))
    month = int(context.arg('month', curtime[1]))
    return year, month, calendar.monthcalendar(year, month)
```

Example 1

Render Methods

- Render methods return values to insert data into Nevow's lightweight DOM

```python
def render_month(self, context, data):
    year, month, weeksAndDays = data

    weekPattern = context.patternGenerator('calendarWeek')
    dayPattern = context.with(weekPattern()).patternGenerator('calendarDay')

    calendarBody = []
    for week in weeksAndDays:
        currentWeek = weekPattern().clear()
        calendarBody.append(currentWeek)
        for day in week:
            currentDay = dayPattern().clear()
            if day != 0:
                currentDay.children.append(str(day))
            currentWeek.children.append(currentDay)

    context.fillSlots('label', "%s %s" % (calendar.month_name[month], year))
    context.fillSlots('calendarBody', calendarBody)
    return context.tag
```

Example 1

Integrating

- We are now going to use an external data source to make our application more useful
- We will modify ScheduleRoot.__init__ to take an iCal entries instance

```
def __init__(self, calendarEntries):
    self.calendarEntries = calendarEntries
    super(ScheduleRoot, self).__init__()
```

Example 2

Modified Renderer

- We then render events in the calendar

```
calendarBody = []
for week in weeksAndDays:
    currentWeek = weekPattern().clear()
    calendarBody.append(currentWeek)
    for day in week:
        if day != 0:
            events = self.calendarEntries.eventsFor(date(year, month, day))
        else:
            events = []
        currentDay = dayPattern(
            render=self.render_day, data=(year, month, day, events))
        currentWeek.children.append(currentDay)
```

- We delegate to “render_day” and pass additional data “events”

Example 2

render_day

- Day detail pages will be located at http://localhost:8080/2004/3/21

```
def render_day(self, context, data):
    year, month, day, events = data
    if events:
        from nevow.url import here
        # Construct URL to child page
        url = here.child(str(year)).child(str(month)).child(str(day))
        eventsDOM = context.onePattern('events').clear()
        eventsDOM.attributes['href'] =url
        eventsDOM.children.append(
            "%s event%s" % (len(events), len(events) > 1 and 's' or ''))
    else:
        eventsDOM = ''

    context.fillSlots('date', day)
    context.fillSlots('events', eventsDOM)
    return context.tag
```

Example 2

URL Traversal

- Accessing /2003/3/21 calls locateChild

```
def locateChild(self, request, childSegments):
    if childSegments == ('',):
        ## We are looking for the root of the site, aka /
        return self, ()
    try:
        year, month, date = map(int, childSegments)
        import day
        return day.DayDetail(self.calendarEntries, year, month, date), ()
    except ValueError:
        # If the url doesn't consist of a tuple of year, month, day, or the segments
        # are not integers, then we render a 404 page.
        return rend.NotFound
```

stan

- Nevow does not use W3C DOM
  - Uses a Python DOM named ‘stan’
- Stan DOM uses basic Python types:
  - Strings
  - Lists
  - Extensible using Adapters

render methods & stan

- Return value of a render_ method is stan

```
def render_date(self, context, data):
   return "%s %s, %s" % (calendar.month_name[self.month], self.date, self.year)
```

- Return value replaces template input
- You can return almost any Python type
  - A render_ method can even be a generator yielding stan

Example 3

stan tags

- Simple DOM Node replacement:
    - nevow.stan.Tag
- tagName string
- attributes dictionary
- children list

Example 3

XML in Python

```
docFactory = rend.stan(
T.html[
    T.head[
        T.title[
            "Detail for ", render_date]],
    T.body[
        T.a(href=root)["Back"],
        T.h1[
            "Detail for ", render_date],
        T.h2[
            "Events:"],
        render_events]])
```

- Tag.__call__ sets attributes
- Tag.__getitem__ adds children

Example 3

Python view logic

```
from nevow import tags as T

class DayDetail(rend.Page):
    def render_events(self, context, data):
        events = self.calendarEntries.eventsFor(
            date(self.year, self.month, self.date))
        if not events: return "No events yet."
        return T.ol[
            [
                T.li[str(e)]
                for e in events
            ]
        ]
```

```
<ol>
    <li>Full Moon</li>
</ol>
```

Example 4

Formless

- Formless lets you **describe types** using **interfaces**

```
class IEventsAddable(formless.TypedInterface):
    def addEvent(self, description=formless.Text()):
        """Add Event

        Add an event with the given description to this day.
        """
        return IEvent
    addEvent = formless.autocallable(addEvent)
```

Example 4

autocallable methods

```
class DayDetail(rend.Page):
    __implements__ = IEventsAddable, rend.Page.__implements__

    def addEvent(self, description):
        newEvent = iCal.ICalEvent()
        newEvent.summary = description
        newEvent.startDate = date(self.year, self.month, self.date)
        self.calendarEntries.events.append(newEvent)
```

- Can be called automatically

```
<form action="freeform_post!!addEvent" method="POST">
    <input type="text" name="description" />
    <input type="submit" />
</form>
```

Example 4

freeform

- Freeform gives you forms free

```
   docFactory = rend.stan(
   T.html[
...
       T.body[
...
           T.p[
               freeform.renderForms()]]]])
```

- Also useful for rendering custom forms

- binding
- action
- argument
- value
- error

Example 5

livepage

- Out of band events
- Client to Server (JS to Python)
- Server to Client (Python to JS)

```
def render_inputBox(self, ctx, data):
    def newValue(client, newValue):
        print newValue
    return input(onchange=handler(newValue, 'this.value'))

def render_clickableImage(self, ctx, data):
    def clicked(client):
        client.sendScript('You clicked the image!')
    return img(onclick=handler(clicked))
```

Q&A

DIVMOD