Templating for Responsive Images

Abhishek Tiwari

Citation: *A. Tiwari*, "Templating for Responsive Images", Abhishek Tiwari, 2013. doi:10.59350/mc8x5-1ra32

Published on: December 24, 2013

Recently I discussed at length about Responsive Images and Responsive Image as Service.

In this post I will discuss how we can use some effective templating techniques to implement highly maintainable responsive image markup.

Templates express presentation logic of a web application. Templates are quite similar to plain HTML markup but they are re-usable and modular. Most template languages have provision for variables, tags, filters, and inheritance.

In a template, variables are used to render the data in the markup. Tags control logic of the template. Filters modify the variables for display. Inheritance allows child template to inherit base template skeleton and override it.

Django Template

Django is a Model-View-Controller (MVC) framework, or more precisely a Model-Template-View (MTV) framework. In Django , MVC Controller is Django View and MVC View is Django Template. A Django Template renders data passed by Django View.

I will demonstrate use of Django templates to implement responsive image markup. Although this demonstration relies on Django template language other template languages can easily support this kind of approach.

Special markup using Django Template

Following template `main.html` is looping over template variable `images` (a list of `image` objects). This variable `images` was passed by a Django View function to template `main.html`. Inside the loop `main.html` is using a custom Django inclusion tag[1] `show_responsive_image` which basically calls another template `responsive-image.html` and passes `image` object. Please note each `image` object has named attributes which can be accessed when required.

```
<!-- main.html Template-->
<html>
    <header>
        <script src="/static/js/picturefill.js"></script>
    </header>

    <body>
        {{ "{% for image in images " }} %}
            {{ "{% show_responsive_image image " }} %}
        {{ "{% endfor " }} %}
```

[1] Django Custom Template Tags-Inclusion tags

```
    </body>
</html>
```

```
<!-- responsive-image.html v1 Template-->
<img src="{{ "{{ MEDIA_URL " }}}}{{ "{{image.path " }}}}" alt="{{ "{{image
    .title" }}}}">
```

At the moment `responsive-image.html` is just rendering normal `img` elements. ~~~ .html ~~~

This could have been done simply in `main.html` so why another template?

Picturefill

Let's try again, with responsive version of `responsive-image.html` based on Picturefill[2].

```
<!-- responsive-image. v2 Template done with Picturefill-->
<span data-picture data-alt="{{ "{{image.title" }}}}">
        <span data-src="{{ "{{ MEDIA_URL " }}}}small/{{ "{{image.path "
            }}}}"></span>
        <span data-src="{{ "{{ MEDIA_URL " }}}}medium/{{ "{{image.path "
            }}}}"     data-media="(min-width: 400px)"></span>
        <span data-src="{{ "{{ MEDIA_URL " }}}}large/{{ "{{image.path "
            }}}}"      data-media="(min-width: 800px)"></span>
        <span data-src="{{ "{{ MEDIA_URL " }}}}extralarge/{{ "{{image.path
             " }}}}" data-media="(min-width: 1000px)"></span>

        <!-- Fallback content for non-JS browsers. -->
        <noscript>
            <img src="{{ "{{ MEDIA_URL " }}}}medium/{{ "{{image.path "
                }}}}" alt="{{ "{{image.title" }}}}">
        </noscript>
</span>
```

This new template will generate following markup source required by Picturefill[3] to provide responsive images.

```
<!-- responsive-image. v2 Template done with Picturefill-->
<span data-picture data-alt="Profile-1">
        <span data-src="/static/images/small/image1.jpg"></span>
        <span data-src="/static/images/medium/image1.jpg"     data-media="
            (min-width: 400px)"></span>
        <span data-src="/static/images/large/image1.jpg"      data-media="
            (min-width: 800px)"></span>
        <span data-src="/static/images/extralarge/image1.jpg" data-media="
            (min-width: 1000px)"></span>
```

[2] Picturefill
[3] Picturefill

```
        <!-- Fallback content for non-JS browsers. -->
        <noscript>
            <img src="/static/images/medium/image1.jpg" alt="Profile-1">
        </noscript>
</span>
<!-- responsive-image. v2 Template done with Picturefill-->
<span data-picture data-alt="Profile-2">
        <span data-src="/static/images/small/image2.jpg"></span>
        <span data-src="/static/images/medium/image2.jpg"     data-media="
           (min-width: 400px)"></span>
        <span data-src="/static/images/large/image2.jpg"      data-media="
           (min-width: 800px)"></span>
        <span data-src="/static/images/extralarge/image2.jpg" data-media="
           (min-width: 1000px)"></span>

        <!-- Fallback content for non-JS browsers. -->
        <noscript>
            <img src="/static/images/medium/image2.jpg" alt="Profile-2">
        </noscript>
</span>
```

Again actual magic or DOM manipulation will happen `onload` by client-side JavaScript using `Picturefill.js` that is included in our markup header. Assuming if you are using a desktop machine, final DOM markup will look like (inserted `img`),

```
<!-- responsive-image. v2 Template done with Picturefill-->
<span data-picture data-alt="Profile-1">
        <span data-src="/static/images/small/image1.jpg"></span>
        <span data-src="/static/images/medium/image1.jpg"     data-media="
           (min-width: 400px)"></span>
        <span data-src="/static/images/large/image1.jpg"      data-media="
           (min-width: 800px)"></span>
        <span data-src="/static/images/extralarge/image1.jpg" data-media="
           (min-width: 1000px)">
            <img src="/static/images/extralarge/image1.jpg" alt="Profile-1
               ">
        </span>

        <!-- Fallback content for non-JS browsers. -->
        <noscript>
            <img src="/static/images/medium/image1.jpg" alt="Profile-1">
        </noscript>
</span>
<!-- responsive-image. v2 Template done with Picturefill-->
<span data-picture data-alt="Profile-2">
        <span data-src="/static/images/small/image2.jpg"></span>
        <span data-src="/static/images/medium/image2.jpg"     data-media="
           (min-width: 400px)"></span>
        <span data-src="/static/images/large/image2.jpg"      data-media="
           (min-width: 800px)"></span>
```

```
        <span data-src="/static/images/extralarge/image2.jpg" data-media="
            (min-width: 1000px)">
            <img src="/static/images/extralarge/image1.jpg" alt="Profile-1
                ">
        </span>

        <!-- Fallback content for non-JS browsers. -->
        <noscript>
            <img src="/static/images/medium/image2.jpg" alt="Profile-2">
        </noscript>
</span>
```

Imager.js

Let say after sometime you decided to replace Picturefill[4] with Imager.js[5]. This requires following changes,

1. User `imager.js` in header instead of `picturefill.js`
2. Add JavaScript code to interpolate width to a string
3. Update the responsive image tag for our template

```
<!-- main.html Template-->
<html>
    <header>
        <script src="/static/js/imager.js"></script>
        <!-- Interpolate {width} to a string -->
        <script>
            new Imager({
                availableWidths: {
                    200: 'small',
                    320: 'medium',
                    640: 'large',
                   1024: 'extralarge'
                }
            });
        </script>
    </header>
    <body>
        {{ "{% for image in images " }} %}
            {{ "{% show_responsive_image image " }} %}
        {{ "{% endfor " }} %}
    </body>
</html>
```

```
<!-- responsive-image. v3 Template done with Imager.js -->
```

[4] Picturefill
[5] BBC-News / Imager.js

```
<div data-src="{{ "{{ MEDIA_URL " }}}}{width}/{{ "{{image.path " }}}}"
    data-alt="{{ "{{image.title" }}}}"></div>
<!-- Fallback content for non-JS browsers. -->
<noscript>
    <img src="{{ "{{ MEDIA_URL " }}}}medium/{{ "{{image.path " }}}}" alt="
        {{ "{{image.title" }}}}">
</noscript>
```

After above three changes and assuming if you are using a desktop machine, final DOM markup will look like,

```
<img src="/static/images/large/image1.jpg" data-src="/static/images/{width
    }/image1.jpg" class="image-replace" alt="Profile-1">
<img src="/static/images/large/image2.jpg" data-src="/static/images/{width
    }/image2.jpg" class="image-replace" alt="Profile-2">
```

Final Words

Here I briefly covered, some clever use of Django template language to write highly maintainable responsive image markup. Although I have not tried this approach with other template languages, I am quite sure this will work with others as well.