Announcing org-webring

Brett Gilio

<2020-08-20 Thu 21:55>

Contents

An update to this article is available here.

Special appreciation to all contributors: Jamie Beardslee, Amin Bandali, Ivan Sokolov, and Alexandru-Sergiu Marton. Thank you to all those who gave review of the Emacs Lisp used in the project, and Drew DeVault whose `openring`[1] served as a useful template.

[1] `https://git.sr.ht/~sircmpwn/openring`

TL;DR

Org-webring is an alternative implementation of a feed-based webring, taking inspiration from openring by Drew DeVault. Intended to integrate with Org-based websites and blogs (either directly, or indirectly, as in ox-hugo), it will fetch a given list of web feed files and correctly parse and format the elements to be displayed for sharing.

---

1 Introduction

There are certain elements from the "old web" that I feel many of us long for, and miss dearly. Personally, it seems as the internet ages we become more disparate and disconnected from each other. Consistently, we are faced with the reality that while the internet has provided us absolutely remarkable advancements, it has also plagued us with isolation.

One element from the web as a whole which I find most fascinating is blogging. Blogs are some of the most adventurous and accessible modes of detailing thoughts and expressions. I love them! A blog can be fictional, personal, political, technical, or whatever you want it to be. With the abundance of different ways to create a blog, it is easier than ever to get started. While I tend to read mostly technical blogs, especially in content areas I am most interested, the plethora of different mediums engaged in blogging are endless.

Blogging, while also a mode of expression, is also a mode of sharing with others. Something that I find particularly rewarding is discovering a particular blog, or blogpost that details something very exciting. I find *sharing* these entries rather exciting as well, especially to like-minded friends and colleagues.

A blog by Drew DeVault[2] had an especially interesting feature, he was sharing recent posts from blogs he was following! Using a piece of software he had written, called `openring`, he had taken the old and often forgotten concept of a webring and made sharing a generated list of blog entries accessible to his audience!

There are many things that I feel `openring` does well, and while I have never used it there are friends of mine who have. They have positive things to say about the software and its apparent simplicity. While I am not a fan of the implementation language chosen for the project, I will easily concede

[2] `https://drewdevault.com/`

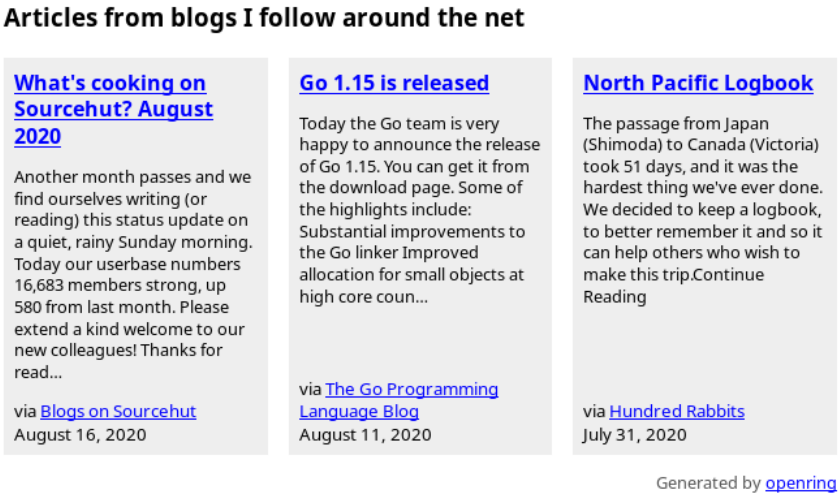

Figure 1: Screenshot of Drew DeVault's openring feed.

that Drew has created something genuine for both himself, and his audience to use.

2 An Org implementation

While Drew's `openring` is quite nice, there were a few things about it that I do not prefer. For example you will need the Go language toolchain to be able to utilize this generator. While that may not seem like a burden to some, it was not preferred for myself and others whom I discussed this with.

Having Go as an extra toolchain for a simple website functionality seemed to be a persistent pain for those of us who generated our websites and blogs statically using tools like Org, Haunt, M4, or Hugo (although Hugo, itself, is implemented in Go). One of the primary benefits of implementations in Org or Haunt was the ease of extensibility. You can easily write what is missing and evaluate the dynamic Emacs Lisp or Scheme to provide the functionality, all out-of-the-box!

With this in mind, some friends and I set out to take the idea of `openring` and reimplement it ourselves, without creating a direct port. Originally this project was called `org-blogfeed`, but was quickly renamed to `org-webring` at the suggestion of Amin Bandali. The implementation of `org-webring` fixes itself around Emacs Lisp (the implementation language of Org), and the Org format itself! Suddenly, we were able to replicate the advantages of

having a sharable blog feed from parsed and generated Org files!

The benefits also extended by simply using the platform of Org. Because Org is so broad in its appeal and approach, people using Hugo could take the `org-webring` implementation and use `ox-hugo` in order to have the result on their Hugo-based webpages.

Below is a demonstration of `org-webring`, using my RSS feed as the sole input while demonstrating some of the optional customizable features:

```
(let ((org-webring-items-total 3) ; default 3.
      (org-webring-items-per-source 3) ; default 1.
      (org-webring-header "Posts from other blogs I follow...")
      (org-webring-display-generation-time t)
      (org-webring-urls '("https://brettgilio.com/rss.xml")))
  (org-webring-generate-webring))
```

2.1 Implementation

Visually, `openring` and `org-webring` have many similarities. This was done on purpose, as our default stylesheet was taken from the `openring` implementation on Drew DeVault's website. It is nice, plain, and easily customizable. There are a few differences, however.

Disregarding the obvious alignment choices (left versus right), we also fold the flex boxes for better visual support on mobile devices, like so:

```
@media screen and (max-width: 600px) {
    .org-webring .org-webring-articles {
display: flex;
flex-direction: column;
    }
}
@media screen and (max-width: 600px) {
    .org-webring .org-webring-article {
flex: 1 1 0;
display: flex;
flex-direction: column;
margin: 0 0.5rem;
margin-bottom: 0.5rem;
padding: 0.5rem;
border: 0.1rem black solid;
border-color: black;
```

```
    }
}
```

Internally, there are many differences barring `org-webring` from being considered a port in the legal sense. Our implementation does not derive any programming choices from `openring`. This comes with some benefits, as well as some work-in-progress decisions to consider for future.

2.1.1 Sanitization

At present, the sanitizer for `org-webring` is not as strict, nor as elegant as the implementation in Go. In `openring`, Drew opted to use a Go sanitizer called `bluemonday`[3].

A redux of the Go implementation is as follows[4]:

```
policy := bluemonday.StrictPolicy()

summary := runewidth.Truncate(
    policy.Sanitize(raw_summary), *summaryLen, "...")
```

Where as the redux sanitization in Emacs Lisp is as follows[5]:

```
(let ((desc-sanitized
       (with-temp-buffer
 (insert (org-webring--feed-text-prop item 'description))
 (apply #'concat (dom-strings
  (libxml-parse-html-region (point-min)
    (point-max)))))))
  `(div :class "org-webring-article"
(p :class "org-webring-article-summary"
   ,(org-webring--string-truncate
     org-webring-description-max-length
     desc-sanitized
     "..."))))
```

The sanitizer in the `org-webring` implementation is basically taking an apply method, and removing all of the known DOM elements defined in the

[3] `https://github.com/microcosm-cc/bluemonday`

[4] `https://git.sr.ht/~sircmpwn/openring/tree/master/openring.go`

[5] `https://git.sr.ht/~brettgilio/org-webring/tree/master/org-webring.el`

`dom` GNU Emacs library[6] to return a sanitized string. This implementation method is almost assuredly not as precise as what is offered in `bluemonday`, however the method is taking the XML elements directly and basically doing the same job.

2.1.2 Modification

There are similarities and differences here. For example, in order to appropriately modify many parts of `openring`, one must first modify the `in.html` file. Many of the customizations are trivial, of course but the result could perhaps be cleaner by simply setting those customizations directly in the Emacs Lisp environment. Touching the HTML file directly seems like less of a fool-proof option, in my opinion.

In `openring`, the articles get generated using the range method, appropriately assigning the number of articles you wish to be displayed to the `in.html` file, and then producing a result with all of your formatted expectations (header, title, date string, attribution)[7]:

```
<section class="webring">
  <h3>Articles from blogs I follow around the net</h3>
  <section class="articles">
    {{range .Articles}}
    <div class="article">
      <h4 class="title">
        <a href="{{.Link}}" target="_blank" rel="noopener">{{.Title}}</a>
      </h4>
      <p class="summary">{{.Summary}}</p>
      <small class="source">
        via <a href="{{.SourceLink}}">{{.SourceTitle}}</a>
      </small>
      <small class="date">{{.Date | datef "January 2, 2006"}}</small>
    </div>
    {{end}}
  </section>
  <p class="attribution">
    Generated by
    <a href="https://git.sr.ht/~sircmpwn/openring">openring</a>
```

[6] `https://www.gnu.org/software/emacs/manual/html_node/elisp/Document-Object-Model.html`

[7] `https://git.sr.ht/~sircmpwn/openring/tree/master/in.html`

```
  </p>
</section>
```

In `org-webring`, these parameters can be evaluated on-the-fly without recompilation using the `defcustom/setq` binding methods.

```
(defcustom org-webring-items-total 3
  "The total number of items that should be in the webring."
  :group 'org-webring
  :type 'integer)

(defcustom org-webring-items-per-source 1
  "How many items should be extracted from each RSS feed."
  :group 'org-webring
  :type 'integer)

(defcustom org-webring-header "Posts from other blogs I follow..."
  "The text of the webring header."
  :group 'org-webring
  :type 'string)

(defcustom org-webring-attribution
  '("org-webring" . "https://sr.ht/~brettgilio/org-webring")
  "Association connecting the name of the program used to
generate the webring to its URL to be displayed at the bottom
of the webring."
  :group 'org-webring
  :type '(cons (string :tag "Name")
               (string :tag "URL")))

(defcustom org-webring-urls '()
  "The links for the RSS feeds to be scraped for items."
  :group 'org-webring
  :type '(repeat string))

(defcustom org-webring-description-max-length 512
  "The maximum length of an item's description."
  :group 'org-webring
  :type 'integer)
```

```
(defcustom org-webring-timestamp-feed-format "%a, %d %b %Y"
  "The format string for the publish dates of feed items. Uses
the same '%'-sequences as `format-time-string'."
  :group 'org-webring
  :type 'string)

(defcustom org-webring-timestamp-generate-format "%a, %d %b %Y -- %R"
  "The format string for the generation of the webring. Uses
the same '%'-sequences as `format-time-string'."
  :group 'org-webring
  :type 'string)
```

3 I have the choice, which one should I use?

Perhaps surprisingly to some, I think this is a personal matter. What you choose should be to serve your own best interest as a programmer or somebody who runs a website. When using `openring`, you must use the Go toolchain. The benefit here is it is quite portable, however Emacs Lisp comes with all Emacs environments and is almost equally as portable. It really comes down to whether *you* like Org (and Emacs, by extension) or not. You could, of course, use Org and `org-webring` for *just* the webring and include the result of the HTML file in any HTML-based result you want, but that takes the fun and integration out of the equation.

In my opinion, you should choose the one you feel has the better toolchain, methods of using it, and will be developed in a direction consistent with your values. Both are free software, so at worst-case you may find yourself trying both! They are both decent pieces of software!

4 Next steps

~~There are a few things we wish to tackle in the future with `org-webring`. Currently the software only supports RSS feeds. While I am partial to RSS feeds as I feel the XML format is superior, there will be a felt gap for some users who wish to have support for ATOM. Currently, without official implementation, you can use an ATOM->RSS converter and host the RSS content yourself but that is cumbersome thus warranting a better solution.~~

- DONE: ~~Support ATOM feeds.~~

In the future, we will also want to consider a more strict and distinct sanitizer for XML/ATOM parsing. As it stands, the sanitizer is effective but it is not elegant.

- TODO: Consider supporting more strict string sanitization.

We are also considering making it possible to use Emacs Lisp to generate a CSS stylesheet for inclusion in the website. One of my griefs with `openring` was that you had to touch the `in.html` file, and while `org-webring` does spare you from this task, you are still required to touch the `org-webring.css` file if you wish to make style modifications.

- TODO: generate CSS from defcustoms

It may be possible to fetch RSS (and eventually ATOM) feeds asynchronously, but currently the synchronous/asynchronous private functions used in GNU Emacs `url-insert-file-contents` public function is not *truly* synchronous nor asynchronous (in the conventional sense). So in order to appropriately apply such a change, we would need to first discuss possibilities with the GNU Emacs maintainers/community, and see about implementing our own way of doing this if needed.

- TODO: Consider implementing generic (async) fetcher

Most projects use the glorious `texinfo` package to generate and aid in distribution of documentation. Currently, we are relying on our ORG->MD parser-generator to supply the functionality of easily accessible web documentation. Eventually we need to deprecte our current method and drop-in a `texinfo` replacement.

- TODO: Get documentation from TEXINFO

Lastly, as GNU Emacs is slowly and incrementally switching to default lexical binding, we should check for possible regressions and make pre-empt this by making it a default for the package.

- TODO: add and check lexical scoping

I am sure there are more hidden bugs, though we have taken a lot of effort (as well as trial and error) to work around any which we have noticed.

5 Where to get it

Currently you will have to do a git checkout to acquire `org-webring`.

```
$ git clone https://git.sr.ht/~brettgilio/org-webring
```

There are plans to add this package to GNU Guix, and eventually consider attributing it to ELPA.

5.1 How to contribute

Please subscribe to the project mailing list and partake in discussion, as well as send git formatted patches (more on this in the README).

Have a response?

Responses and discussion pertaining to any of the blog entries on my website are welcome! Start a discussion on the mailing list by sending an email to ~brettgilio/blog-discussion@lists.sr.ht.

Errata:

- *<2020-08-23 Sun 15:28>* Fix small typo. Add footnote to `openring`.
- *<2020-08-29 Sat 21:40>* ATOM feeds have been implemented.
- *<2020-10-07 Wed 16:07>* Mention status update.
