Mindtree

Welcome to possible

WHITE PAPER

Customized web crawling using Heritrix

Summary

Quite a few businesses today need to crawl other websites and extract information. For example, deals website crawl prospective websites and collate all the deals available, price comparison websites crawl other websites to collect pricing details, social sentiment analysis websites crawl the web to discover opinions about certain brands and then extract information and so on.

There are a range of web crawlers available in the market today. These products do an all-out crawling i.e. they start with the home page, extract all links in the home page and then inturn crawl those pages. This continues till all the links within the current website are done. This does not work in some scenarios because businesses just want to extract some selective information from some selective pages. Identification of the required pages and extraction of selective content are the challenging tasks.

In this approach paper we will delve deeper into multiple aspects of customized web crawling and list the feature requirement from the crawler. We will then do gap analysis of features required and available in out of the box Heritrix crawler. After that, we will detail out the possible Design and Architecture changes in Heritrix to enable customized web crawling. It aims to educate the user of the problem and suggest one possible way of solving it. In order to understand the problem and the suggested solution in this paper, the reader is expected to have some information about web crawlers in general and know Heritrix in particular. This paper does provide references to external useful reading material as suitable.

Contents

Problem statement

Let's understand the need for customized crawling using an example. The business requirement is to get all the product information from a fictitious e-commerce website called www.ecommercewebsite.com. Here are the few detailed requirements / challenges that come under three broad categories:

1. The web crawler should collect information about all products on this website
2. Product information will include
   - Product name
   - Product short description
   - Product long description
   - Download product images – Thumbnail and others
   - Product pricing
   - Product category – For example women's shoes
   - Any promotions deals like – USD five off, buy three get one free and so on
3. The crawler should then pass on this information to another system over web service calls

Now let's superimpose these to standard out of the box features for any web crawler. We will pivot all our discussion around Heritrix, but majority of it will be true for other crawlers too.

The block diagram of the end system will be as depicted in figure 1.

Heritrix is the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project. For more details, please visit http://crawler.archive.org/index.html. The business requirements translate to following technical requirements, which are more focused with customized crawling and not with basic crawling (for more details refer Reference[1]). In the below section we will look at the technical requirement and also map those on to Heritrix

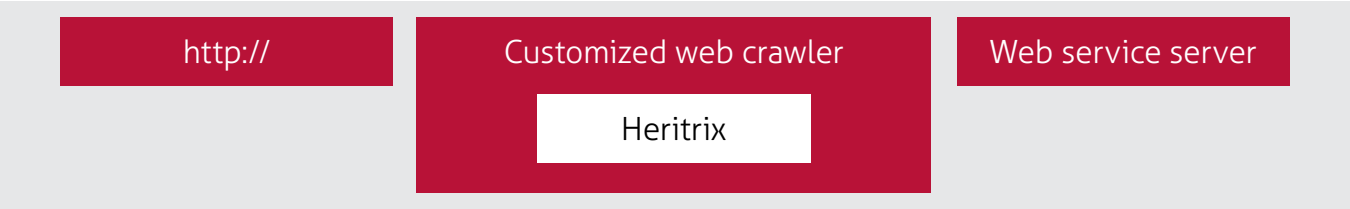

Figure 1 – Customized web crawling using Heritrix system block diagram

Technical requirement	OOTB Heritrix feature	Identified gap
Crawler should be scalable and should be able to crawl multiple websites at once.	Heritrix crawler has configurable toe threads and can be configured to crawl numerous websites in parallel. Default to threads is 50.	None.
Crawler should be configurable to crawl only certain URL patterns and ignore the rest. This can be used to do focused crawling.	Heritrix does provide a bit of URL filtering based on regular expressions, but it's not very effective. It is buggy and is very complex.	Home grown solution needs to be put in to have effective URL filtering.
Crawler should identify duplicate URL's and not recrawl them. Additional requirement is that dedupe should not be only with URL string comparison as at times the URL parameters sequence is changed but its a duplicate logical URL.	Basic URL string dedupe is there, but the advanced requirement is not fulfilled.	Home grown solution needs to be put in.

Crawler should be able to extract selective information from the web page and pass it on to a web service.	No such feature exists in Heritrix. At best Heritrix can be used to dump the web page content and then write additional code to do the extraction and then web service posting. But this results in time lag for writing the page and then reading it.	Home grown solution needs to be put in.
The sequence of the parameters to the web service and how and where to pick them from the web page should be configurable.	No such feature available in Heritrix.	Home grown solution needs to be put in.
For web service parameters, string manipulation such as concatenation should be possible.	No such feature available in Heritrix.	Home grown solution needs to be put in.
Crawler should have memory across pages. For example the category name is available in the previous page but needs to be remembered in the product catalog page and then product details page.	Heritrix is complete stateless and has no memory across pages.	Home grown solution needs to be put in.
Crawler should be able to download the product images and pass those to the web service.	Heritrix can be configured to download images, but there is no way to remember which product the image is for. Also passing on the image to the web services feature is not .	Home grown solution requirement.

Based on the above table, we conclude that although Heritrix is a very good web crawler, it does not satisfy quite a few requirements for customized web crawling. Hence an add-on wrapper solution needs to be built around Heritrix to make it satisfy the above business and technical requirements.

In further sections we will study in detail one such suggested way to do the same.

High level solution

We will start with a brief description of Heritrix and then delve deeper on the suggested solution. Below is the Heritrix basic architecture diagram.

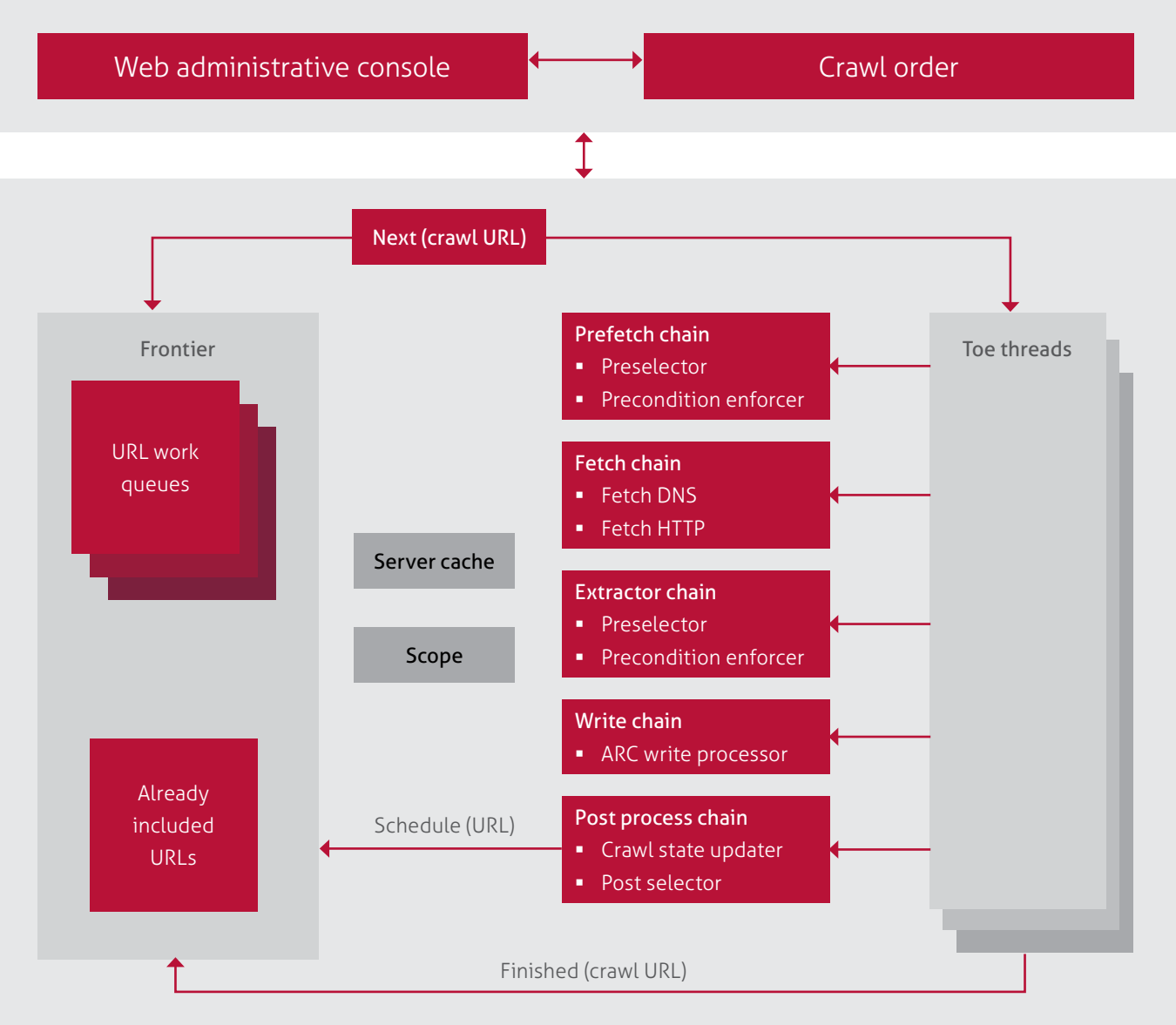

Figure – 2 Heritrix basic architecture – Reference [2]

Please refer to items in reference[1, 2 and 3] for detailed description of the above architecture diagram in figure 2 and its sub components. We are mainly interested in the processing chain components. These components are called in a predefined sequence to process each URL. Below is a brief description of each sub component in the chain.

- **Prefetch chain** – responsible for investigating if the URL could be crawled at this point. That includes checking if all preconditions are met (DNS-lookup, fetching robots, text, authentication)
- **Fetch chain** – processors in this chain are responsible for getting the data from the remote server
- **Extractor chain** – Process the HTML page. Typical functionality is to fetch new links from the web page and feed that back to the frontier
- **Write chain** – Writing data to the archive
- **Postprocess chain** – Do 'clean up' and return control to the frontier

This paper suggests putting our home grown logic in the extractor chain. We will write our home grown extractor which will carry out the custom tasks. Below is the block diagram of our custom extractor (figure 3).

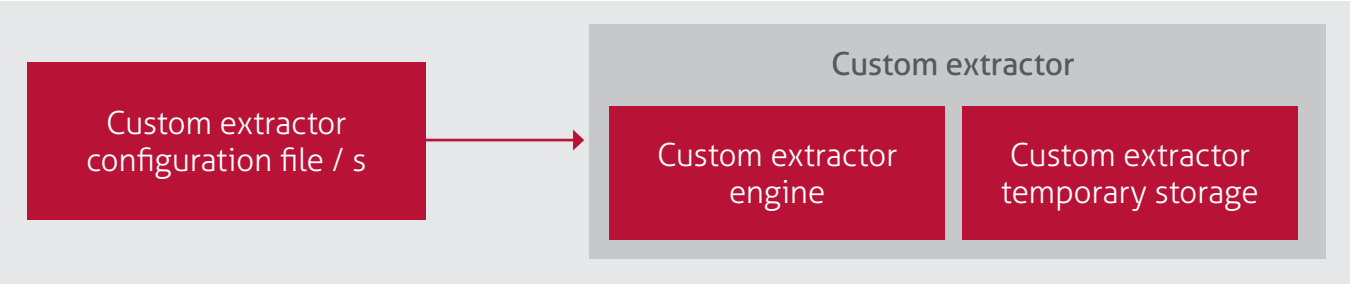

Figure 3 – Custom extractor block diagram

A custom extractor will extend the basic Heritrix extractor interface and will have add-on logic as required. In addition it will have a temporary storage which will store data which needs to be carried across page crawls. This custom extractor will be configured using a configuration file. The site specific configuration will be stored in this file and will be used at run time.

Detailed solution

We will first discuss the overall design and settings for Heritrix and then delve deeper in the custom extractor.

Overall settings for Heritrix

1. Heritrix's job is to have the overall settings and have seed URL's for the websites to be crawled. One job per website can be created or multiple websites can also be combined in a job. Seed URL could be the home page or also the sub page if only certain data needs to be extracted. For example, if only products in certain categories are to be extracted then seed URL could be that category's home page.
2. Create a custom extractor and register it in Heritrix. The job should have this extractor in the processing chain to be configured using the admin console.
3. All across the website, settings like number of toe threads, time out values and so on are to be done at job level. No website specific settings are to be done at job level. This way same Heritrix job file (order.xml) can be used across website crawls.
4. Rest use Heritrix standard job settings.

Apart from the custom extractor, no other code changes are required in Heritrix. The below design can be applied to all types of extractors. For example, HTMLExtractor, XMLExtractor, JSONExtractor and so on. Ideally the code should be written as stand-alone utilities which can be called from above listed extractors.

Custom extractor design

1. Custom extractor engine – This will extend the default Heritrix extractor and will overwrite methods as required. This will be the heart of the system and will read the configuration file and crawl websites as instructed in the configuration. It will also interface with a temporary storage. The features of the engine can be deduced from the configuration file settings. The engine should orchestrate and implement the features described in the configuration file.
2. Custom extractor temporary storage – This is in principal a hash map which will store certain values which need to be remembered across pages. There will be a page which stores in the hash map (source page) and there will be another page which will extract this information and use it (target page) in some way. The source page will store the information in the storage with the key as the 'Target page URL'_'key name', for example www.theecommercestore.com\product1234_categoryId wherein categoryId is the key prefixed by the URL. The target page will use its own URL and the known key name to extract the information and use it as suitable.
3. Image extraction – Additional logic can be put in the engine to extract the images and feed them to the web services. Even the frontier can be used to do the extraction of these images or custom code can be used for the same. Custom code will be preferred as it can be called from the extractor itself and no more hooks needs to put in to Heritrix.
4. Configuration file – This file will orchestrate the engine and carry out the crawl for each website. The file will have sections for each website. Below are few suggested snippets of the configuration file which we will discuss further.

Non-leaf pages

Below snippet is for the non-leaf pages i.e. pages for which further page crawling is required. For example. in the e-commerce context the product detail page is the leaf page and anything before it in the hierarchy is the non-leaf page.

1. For every seed URL, Heritrix will fetch the content and will fetch all the URL's from the webpage. Our custom code needs to filter out which URL's we add back to the frontier, which ones we neglect, the ones we add back to frontier, what extra processing we do.
2. As shown in the above config file, each URL fetched will be put to test with a regular expression. Only if the regex matches, the URL will be fed back to frontier. Else it will be neglected. This will filter out the unrequired pages.
3. For the pages we want to crawl, there might be some data which needs to page from this source page to the target page. For example, this page could be the catalog page for a certain category and this category name / id needs to be passed to the product page because it needs to be in the web service. This data could come either from the source page URL (URL parameters) or from the html page. Above snippet shows the two cases.
   - The first case has xpath='url' added the parameter name as categoryId. This will extract this parameter from the source URL and add it to the temporary storage with the 'target url_parameter name' format.
   - The second case is when the data is to be picked from the HTML content using xpath. The system will extract the content using xpath. In case the data needs to be further scrubbed then addition regex can be used. For example, the deals data could come in various formats like 'buy two get one free', USD five off on purchase above USD 50 and so on. For such cases additional data filters can be provided which will be in a particular sequence. The engine will start matching from the top and whichever pattern matches, the output format will be used to create the output string and add it to the temporary storage.

```
<url-pattern pattern=".*theecommercestore.*categoryId=.*"/>
<data-extraction>
  <data-xpath-simple>
    <data-xpath xpath="url" stringObjectName="categoryId"/>
    <data-xpath xpath="//*[@id='product-detail']//*[@class='value price']/text()" stringObjectName="itemSalePrice">
      <data-filter data-pattern=".*\$([\d]+\.[\d]+).*" data-output="${1}"/>
      <data-filter data-pattern=".*\$([\d]+).*" data-output="${1}"/>
      <data-filter data-pattern="(.*)" data-output="${1}"/>
    </data-xpath>
 </data-xpath-simple>
</data-extraction>
```

Leaf pages

Below snippet is for leaf pages, i.e. pages which extract actual information and pass it using web service.

- The URL regex is described above.
- The data extraction has a new attribute called 'use-for-webservice', which will tell the system to extract the data-xpath elements and use it for a webservice call..
- The data-xpath are similar to the above defined. Additionally there is one new type of xpath called 'memory'. This will tell the system to pick up that variable from the extractor hash map.
- At the end, the webservice name is specified. System will call this webservice with the parameters extracted in the data extraction. The sequence and source of the parameters is defined in the data extraction.
- The above design elements will permit us to fulfill all the business and technical requirements stated in section 1. These changes will successfully customize Heritrix with minimal changes to core engine and yet fulfill all requirements.

```
<url-pattern pattern=".*productId.*"/>
<data-extraction>
  <data-xpath-simple use-for-webservice="true">
    <data-xpath xpath="//*[@id='product-detail']//*[@class='value price']/text()" stringObjectName="itemSalePrice">
      <data-filter data-pattern=".*\$([\d]+\.[\d]+).*" data-output="${1}"/>
      <data-filter data-pattern=".*\$([\d]+).*" data-output="${1}"/>
      <data-filter data-pattern="(.*)" data-output="${1}"/>
    </data-xpath>
    <data-xpath xpath="//*[@id='product-detail']/*[@class='fn']/text()" stringObjectName="itemName">
    </data-xpath>
    <data-xpath xpath="memory" stringObjectName="itemProductId">
    </data-xpath>
  </data-xpath-simple>
</data-extraction>
<call-webservice name="serviceName"/>
```

Business benefits

Such a tool can be used by businesses to extract selective info from other websites and use this aggregated information for multiple purposes. For example:

- Aggregate deals catalog – Deals websites crawl other websites and extract product and deals information and serve it to end customers as an aggregate deals catalog.
- Mine social sentiment – This engine can be used for the discovery part of social mining to discover and extract opinion data from the web.

Businesses can then use this information in multiple ways and benefit from them.

Summary

This paper has presented one solution for customized web crawling using Heritrix. It starts with appreciating the problem and then provides details on the design of the possible changes required for the same. This should give a jump start to anybody who wants to understand the concept of customized web crawling and implement the same.

References:

Item	Description
Heritrix home page	http://crawler.archive.org/index.html
Adaptive_Revisiting_with_Heritrix_-_Thesis	http://skemman.is/stream/get/1946/2071/6500/1/Adaptive_Revisiting_with_Heritrix_-_Thesis.pdf
Heritrix developer manual	http://crawler.archive.org/articles/developer_manual/overview.html

About the author:

Deepak Garg is a Technical Director at Mindtree and is a part of the Solutioning and Consulting Lead Engagements team. He has more than 14 years of rich experience in the IT industry across various domains like telecom / new media / eCommerce / retail / social and so on. Deepak has worked in a wide range of roles in the said domains in positions including Senior Architect, Sales, Pre-Sales, Consulting, Solutioning and Delivery.