mymodern

a modular and thoroughly modern Crystal program

B Traven

apropos.dev

mymodern

command line program

- scrapes web pages at 1, 2 or 3 public libraries' web sites.
- lists checked out books and books on hold ready for pickup
- parses the web pages using CSS selector rules

It’s written in Crystal

The Crystal language

- is a compiled Ruby
- with a way of specifying concurrency using communicating sequential processes **(CSP)**, as in Go

Using CSPs to manage concurrency

A simple fragment, where we fetch exactly two webpages, and have hard coded names for them.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

Create two channels, passing them to two spawned tasks (which don’t immediately run), ask to receive a web page from the first channel.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

The program ‘blocks’ at the first receive, and yields to any unfinished task, the first get_page says I haven’t run yet, and does an http get…

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

The first http get is blocked, waiting on the internet – so the program yields to the second get_page that was spawned,

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

The second get_page does a get a fraction of a second after the first one. Now we have two http gets both waiting on a response from the internet.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

One get_page gets a response before the other and sends the webpage on the channel it was given. Let’s say it was the on hold web page.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

Recall we are also waiting to receive a webpage on the checked out webpage channel. If a page was sent on the on hold channel, we still are.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

half a tick later we get the checked out page from the internet, it’s sent on its channel, and that’s the channel we were waiting to receive on.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

Execution moves on to the 2nd receive, and in this scenario, the on hold page has been previously sent, and is immediately received.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

Execution of one get_page was started immediately **after** the other, with each waiting for a webpage during the same time interval.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

With execution of each get_page interleaved with the other, the one starting after the other, we have concurrency, but not parallelism.

```
# create a channel for each page
checked_out_webpage_channel = Channel(Webpage).new
on_hold_webpage_channel     = Channel(Webpage).new

# Spawn
spawn get_page(checked_out_webpage_channel, cookies, data_param[:checked_out_url])
spawn get_page(on_hold_webpage_channel,     cookies, data_param[:holds_url])

# Receive
checked_out_page = checked_out_webpage_channel.receive
on_hold_page = on_hold_webpage_channel.receive

def self.get_page(webpage_result_channel, cookie_headers, the_url)
  # get one of the pages
  the_response = @@http_client.get(the_url, cookie_headers)
  # Send the result on one of the channels
  webpage_result_channel.send the_response.body
end
```

what happens is

“card=123781898958 42&pin=012387”

our library card number and pin is the login form information sent using the http post method.

1. the website says “hey, they knew the pin associated with the card number”

2. the website remembers our card number as a value in the session table with a random number as the look-up key.

3. the website sends a response to our login that returns to us the look-up key as a cookie

we posted, we got a response

The response to our login post is lines of text, a sequence of characters with newlines in it. Everything after the first 2 newline characters in a row is the web page.

The first lines are the head of the response, and one of the headers is the cookie we need to remember.

HTTP::Client.get(the_url, the_cookie)

We can now use the http get method along with the checked out books URL and the cookie we've been given.

The website gets the get request, and

1. looks up our library card number in the session table using the cookie as the key,
2. uses our library card number to find our checked out books, and
3. turns that list of books into a webpage and sends it back to us

without concurrency

A non-concurrent program would **wait** for each post or get request to complete before doing the next.

For 2 websites, where we fetch 2 pages from each one, it’s

- login post, get request, 2nd get request
- login to 2nd website, 3rd get and then 4th get request

waiting on 6 requests in turn.

with concurrent requests

We can post a login request to one website, and without waiting for a response, post a 2nd login to another website.

So the program is waiting in two different places to continue on from as soon as a response is returned from either of our two login requests.

When we see a response to one of our login posts, we do a get request and a fraction of a second later a 2nd get request.

The goal is to use **concurrency** to get pages back to us more quickly.

waiting on a get and a get, and that 2nd cookie

Let’s say the 2nd login response with its cookie is received not long after the first.

As soon as we see it, set up to do 2 concurrent gets to the 2nd website.

Doing get requests one after another is not parallelism, but when **one get request *immediately* follows another**, the program ends up waiting for each of the web pages during the same period of time. Which is concurrency.

waiting on a get and a get, and a get and a get

post, post

then wait … and …

2 cookies come back at around the same time

see 1 cookie, do

get, get (lets say short wait here)

see 2nd cookie do

get, get

wait … and …

(We just did 2 gets **one right after another**,

and after maybe a short interval, did 2 more gets **in quick succession**.)

Now we… wait and…

collect all four get responses at around the same time.

to reiterate

post, post

wait and ...

2 cookies come in at around the same time

get get, maybe short wait, get get

wait and ...

receive four webpages at around the same time

mymodern is somewhat modular

You set up to scrape one or two libraries' websites by writing Crystal code.

The code that calls upon your code doesn't really care about how many modules you've supplied or what you named them.

hennepin.cr

Supply a file like this for each library website you want to do. You implement these 3 methods.

```
module Hennepin
  def self.lib_data
    # returns data in the form of a NamedTuple

  def self.parse_checkedout_page(page)
    # return a sorted array of CheckedOutRecord
    # uses already existing record and comparison definitions

  def self.parse_on_hold_page(page)
    # return a sorted array of OnHoldRecord
```

And whatever other helpers to use when parsing the web pages.

The lib_data method.

The {}’s and what’s inside them is a named tuple that the method returns. The method name and keys are the same in another library’s module, the values are specific to Hennepin.

```
def self.lib_data
  {
    post_data:           HennepinSecrets::POST_DATA,
    post_url:            HENN_BASE_URL + "/user/login?destination=%2F",
    checked_out_url:     HENN_BASE_URL + "/v2/checkedout",
    holds_url:           HENN_BASE_URL + "/v2/holds/ready_for_pickup",
    print_name:          "Hennepin",
    checked_out_fixture: "h_c_v2.html",
    holds_fixture:       "h_h_v2.html",
    trace_name:          "Hennepin",
  }
end
```

The lib_data method.

```
post_data:          HennepinSecrets::POST_DATA,
post_url:           HENN_BASE_URL + "/user/login?destination=%2F",
checked_out_url:    HENN_BASE_URL + "/v2/checkedout",
holds_url:          HENN_BASE_URL + "/v2/holds/ready_for_pickup",
```

Already existing code uses the lib_data information to post the ‘post_data’ to the ‘post_url’, and then get the pages at the ‘checked_out_url’ and ‘holds_url’.

```
login_response =
  @@http_client.post(data_param[:post_url], form: data_param[:post_data])

the_response =
  @@http_client.get(the_url, cookie_headers)
```

In hennepin_secrets.cr

```
module Hennepin
  HennepinSecrets::POST_DATA = "card_number=12345234534567&user_pin=4321"
end
```

Part of the hennepin module is defined in hennepin_secrets.cr, so the library card number and secret pin are in a separate file, as a first idea for keeping them secret.

In hennepin_secrets.cr

```
module Hennepin
  HennepinSecrets::POST_DATA = "card_number=12345234534567&user_pin=4321"
end
```

The string is the login form information for a form with fields named card_number and user_pin.

The string is sent as is, when supplied as an argument to an already written post request.

```
login_response = @@http_client.post(
    data_param[:post_url], form: data_param[:post_data])
```

Specifying POST_DATA constant

```
module Hennepin
  HennepinSecrets::POST_DATA = "card_number=12345234534567&user_pin=4321"
end
```

The card_number and user_pin in the above string are values of name attributes of input tags in the website's login form shown below.

```
<form action=
"https://thelibrary.com/user/login?destination="
method="post">
  <input name="card_number" type="text">
  <input name="user_pin"    type="password" value="">
  <input name="commit"      type="submit"   title="Log In">
```

The Hennepin module

```
module Hennepin  # hennepin_secrets.cr file
  HennepinSecrets::POST_DATA = "card_number=12345234534567&user_pin=4321"
end
```

HennepinSecrets::POST_DATA is mentioned in the more complete definition of the hennepin module in the hennepin.cr file.

```
module Hennepin   # hennepin.cr file
  def self.lib_data
    {
      post_data:     HennepinSecrets::POST_DATA,
      etc, etc...
```

Specifying post_url: value

```
<form action=
"https://thelibrary.com/user/login?destination="
method="post">
  <input name="card_number" type="text">
  <input name="user_pin"    type="password" value="">
  <input name="commit"      type="submit"   title="Log In">
```

The url we post to seems to be the url of the login page or we should say is the value of the action attribute of the form tag or the formaction attribute of the input tag with type=”submit” or a button with a formaction.

The lib_data method.

Besides the post_data and 3 url's in the lib_data method we have

```
def self.lib_data
  {
    ...
    print_name:           "Hennepin",
    checked_out_fixture: "h_c_v2.html",
    holds_fixture:        "h_h_v2.html",
    trace_name:           "Hennepin",
  }
end
```

lib_data print_name:

```
print_name: "The Municipal Library of Kalamazoo Michigan"
```

How you want the library's name to appear in the report.

```
The Municipal Library of Kalamazoo Michigan Books Out

The Astronomer
Tuesday December 04, 2018

The Municipal Library of Kalamazoo Michigan Books on Hold

A History of America in Ten Strikes
Wednesday November 14, 2018
```

lib_data fixtures

```
checked_out_fixture: "h_c_v2.html",
holds_fixture:       "h_h_v2.html",
```

Given the ‘- -mock’ option, the program gets webpages from the disk instead of the internet.

When you write the parse checkedout page and parse on hold page methods, you will probably get the pages in your web browser, save them, and puzzle it out.

This program still has code that refers to the web pages you should (still) have on disk.

lib_data fixtures

```
checked_out_fixture: "h_c_v2.html",
holds_fixture:       "h_h_v2.html",
```

The name of the checked out books web page is the value for the checked_out_fixture key.

And the name of the on hold web page on disk, h_h_v2.html in this example, is the value for the holds_fixture key.

lib_data fixtures

```
checked_out_fixture: "h_c_v2.html",
holds_fixture:       "h_h_v2.html",
```

When you supply the ‘- -mock’ option when you run the program, the program uses a mock http client that associates urls that get web pages with files on the disk.

This is done for you in my_mock_client.cr. The only part that is somewhat hard coded is the directory path to the webpages. Put your pages where the already existing ones are.

lib_data trace_name:

```
trace_name:                     "Hennepin",
```

Given the ‘- -trace’ option, the program prints trace messages that demonstrate concurrency when fetching web pages. This value represents the name of the library in the trace messages.

```
0.021  :  fetch_pair : doing post for Hennepin
0.242  :  fetch_pair : doing post for StPaul
2.067  :  fetch_pair : done with post for StPaul
```

The parse methods

Each of the one or two or three modules for the one or two or three libraries we are scraping not only need a

```
def self.lib_data
```

method, but also this pair of methods

```
def self.parse_checkedout_page(page)

def self.parse_on_hold_page(page)
```

The parse methods

Here are two possible problems with parsing web pages at public library web sites. Or any web page.

Some Web 2.0 XMLHttpRequest process, or whatever the children are using this century, that causes a get of a web page to not return a complete page. You got a page but it doesn’t contain the information you want, presumably it will be filled in later somehow.

The parse methods

You got a page but it doesn’t contain the information you want. Because fancy. Is what I’m positing.

I’m guessing public library web sites are not that cutting edge. If it is, maybe look for a set of web pages that is the previous interface. The classic interface as it were. Or look for a simple ‘print version’ of the webpage that consists of old school tables.

The parse methods

Second problem, just saying, would be…
You got a page but the HTML tags and classes are thoroughly obscured with various and sundry artifacts.

When working on discovering the structure of the page, delete the extra stuff, and pretty print what’s left. The program consumes the large complex thing, but when figuring it out just work with HTML tags and class attributes, as Nature intended.

The parse methods

… when figuring it out just work with HTML tags and class attributes, as Nature intended.

> *“Take a minute to think about it, and then guess,” said the Red Queen. “Meanwhile, we’ll drink your health…”*

The parse methods

Once you've found the parts of the pages that represent the sequence of books in the webpages, just look at the example code.

The two libraries I'm doing have top level parse_checkedout_page(page) methods differing only in their css selector rules, one page has books inside of divs, the other uses table rows.

```
books_out = doc.css("div.cp-checked-out-item").to_a.map do |book_part|
# versus
books_out = doc.css("table tr.patFuncEntry").to_a.map do |book_part|
```

The parse methods

The St. Paul parse_on_hold_page method has a conditional to test for books whose hold status is Ready. (see next slide for comments)

```
def self.parse_on_hold_page(page)
  doc = Myhtml::Parser.new(page)
  books_on_hold =
  doc.css("table tr.patFuncEntry").to_a.compact_map do |book_part|
    td_status = book_part.css("td.patFuncStatus").first
    status = td_status.inner_text.strip
    if status.match(/^Ready/)
      the_title = find_on_hold_title(book_part)
      the_date = find_on_hold_date(status)
      OnHoldRecord.new(the_title, the_date)
    end
  end
  books_on_hold.sort { |a, b| Recs.holds_compare(a, b) }
end
```

```
For on hold books, status might be
          <td class="patFuncStatus"> 16 of 19 holds </td>
We want to select only those like
          <td class="patFuncStatus"> Ready. Must pick up by 10-25-16 </td>
if generates a record when status.match(/^Ready/) is true,
otherwise the if statement produces nil.
compact_map removes nils,
then maps the non-nil elements by evaluating the block
This allows operating only on book_parts that matched /^Ready/

def self.parse_on_hold_page(page)
  doc = Myhtml::Parser.new(page)
  books_on_hold =
  doc.css("table tr.patFuncEntry").to_a.compact_map do |book_part|
    td_status = book_part.css("td.patFuncStatus").first
    status = td_status.inner_text.strip
    if status.match(/^Ready/)
      the_title = find_on_hold_title(book_part)
      the_date = find_on_hold_date(status)
      OnHoldRecord.new(the_title, the_date)
    end
  end
  books_on_hold.sort { |a, b| Recs.holds_compare(a, b) }
end
```

Development

The program calls upon identically named methods provided by one or another library's module.

```
module Hennepin
  def self.lib_data
    # return a NamedTuple

  def self.parse_checkedout_page(page)
    # return a sorted array of CheckedOutRecord
    # the record and comparison definitions are done elsewhere

  def self.parse_on_hold_page(page)
    # return a sorted array of OnHoldRecord
```

Development

```
require "./hennepin"
require "./hennepin_secrets"
require "./stpaul"
require "./stpaul_secrets"

MODULE_NAMES = [Hennepin, StPaul]
```

Code for another library where you're a regular patron would provide the same, and parts of the program that invoke these methods iterate over an array of the module names, and don't hard code the number or names of the modules.

Development

```
require "./hennepin"
require "./hennepin_secrets"
require "./stpaul"
require "./stpaul_secrets"

MODULE_NAMES = [Hennepin, StPaul]
```

You can adapt the modules that define these few methods, name them Springfield and Shelbyville, and require them from a single file which also lists them in an array of module names.

Development

```
require "./hennepin"
require "./hennepin_secrets"
require "./stpaul"
require "./stpaul_secrets"

MODULE_NAMES = [Hennepin, StPaul]
```

The file with content similar to the above must be named ‘module_names.cr’

Development

The already written code has the aforementioned file name in its list of required files.

```
require "http/client"
require "myhtml"
require "./record_types"
require "./print"
require "./my_mock_client"

require "./module_names"
```

Development

In module_names.cr the constant naming the array appears in the already written code.

```
MODULE_NAMES = [Hennepin, StPaul]
```

```
def self.concurrent_network_part
  the_channels = MODULE_NAMES.map do |a_module|
    the_channel = PagesChannel.new
    the_params = a_module.lib_data
    spawn fetch_pair(the_channel, the_params)
    {the_channel, a_module}
  end
  # etc
end
```

Development

So this program can be used to scrape web pages at 1, 2, or 3 libraries, and all things being equal, runs in constant time no matter how many you’re doing.*

* Because concurrency.

here’s a picture of a bunny with a pancake on it’s head