

Project: Youtube Summarizer Server

Downloading YT Transcripts

Arnold Doray - 9 Jan 2024

Discussion

- `*/net` to download synchronously
- `*/json` to easily parse JSON files
- The `arnold/ytt` library for downloading transcripts.



Synchronous HTTP from ***/net**

- **HTTP-GET** (“url” — “s”) for simple synchronous HTTP downloading.
- **HTTP-GET2** (“url” # — #) for downloading with a headers #. This is needed to inject headers eg a secret key. Response is stored under key “Response”.
- You can use special headers to prevent unauthorised access to your server.
- There are also **HTTP-POST** and **HTTP-POST2**
- You need to require **arnold/net**

Setting HTTP Parameters in ***/net**

- You can add parameters prior to HTTP-XXX using **PARAM** (“key” “value” —), instead of setting manually in the URL. These are available on all subsequent HTTP-XXX calls.
- You can manually clear parameters sent using **CLEAR-PARAMS** (—)
- **USER-AGENT** (“s” —) sets the user agent. Some servers require you set this correctly.
- This can be a second layer of protection for your server against DOS attacks: Check for the user agent and disregard if it is not valid.

Example 1: HTTP-GET

```
: main ( -- )  
  
  \ prints out the page.  
  "https://smojo.ai/" http-get .  
  
;
```

- The page is printed out.
- But if there is an error, it will say “null”

Example 2: HTTP-GET2

```
: main ( -- )
    "https://smojo.ai/" # http-get2 { h }
    h . cr
    "Response" h #@ . cr
;
```

- The response hash is printed out first.
- The key/value pairs are entries in the incoming HTTP response, which you can examine.
- Many values are stored in tuples
- The `null` key holds the main HTTP header response (eg 200 ok, 404, etc) This can be useful for diagnostics.
- The actual data is held under the key “Response” This can be null if there is an error.

DEMO:
**Using HTTP-GET and HTTP-
GET2**

Parsing JSON responses with */json

- JSON is a widely used M2M data exchange format, esp in AI/ML.
- The Youtube transcript data is stored in both JSON and XML.
- **JSON>** ("s" — <json>) converts JSON text into a JSON object.
- The JSON object can be a #, a JSON array or a simple Smojo string/integer/real
- JSON arrays can be converted to Smojo tuples using **JSON>ARRAY** (<json> — tuple)
- **>JSON** (<json> — "s") serializes a JSON object back into a string.

Accessing data with */json

- Most JSON documents are structured in a tree format.
- **[JPATH]** (|s —) compiles a JSON Path to access a portion of the document.
- Eg:
- **[JPATH]** name/id/version
- Will access the **name** then **id** then **version** from the input JSON object.
- You can use **selectors** if there are more than one option, eg:
- **[JPATH]** name[3]/id/version
- will select the 4th **name** object from an array. (Index starts with 0)
- You can also use function selectors, eg:
- **[JPATH]** name/id/version[latest] will use the latest word to run processing on the version nodes.

Example 3: **JSON>** and **[JPATH]**

```
require arnold/net
require arnold/json

: main ( -- )

  "https://app.smojo.org/arnold/net.json" http-get 1
  json> [jpath] symbols/http-post . cr 2 3 4
;
```

- Step 1 - **HTTP-GET** is used to download the JSON text
- Step 2 - Convert this into JSON Object using **JSON>**
- Step 3 - Access value of **symbols/http-post** using **[JPATH]**
- Step 4 - Print it.

Example 4: Using selectors

```
require arnold/net  
require arnold/json
```

```
: main ( -- )
```

```
  "https://app.smojo.org/arnold/net.json" http-get  
  json> [jpath] instructions[23] . cr
```

2

3

4

1

- Step 1 - **HTTP-GET** is used to download the JSON text
- Step 2 - Convert this into JSON Object using **JSON>**
- Step 3 - Access 24th instruction using **[JPATH]**
- Step 4 - Print it.

Example 5: Using functional selectors

```
require arnold/net
require arnold/json

: sum ( json-array -- n )
  json>array array>seq
  0 swap [: + ;] reduce
;

: main ( -- )

  "https://app.smojo.org/arnold/net.json" http-get
  json> [jpath] instructions[sum] . cr
;
;
```

- This example is atypical — usually you use a functional selector for selecting an element not for transformation.
- We can write **SUM** outside of the JPATH here since it is the final step the JPATH

DEMO:
Using JSON> and [JPATH]

Downloading YT Transcripts

- You can easily download YT transcripts using `*/net` and `*/json`
- I've translated some opensourced Python code into Smojo:
- <https://github.com/jdepoix/youtube-transcript-api>
- [arnold/ytt](#) (YT Transcripts).
- Have shared the source code on my profile page.

Using **arnold/ytt**

- **ytt.transcript.raw** ("video-id" — "s") grabs the raw XML file.
- **ytt.transcript.timed** ("video-id" — seq-#) grabs the data into a sequence of #, each with **time**, **duration** & **text**.
- **ytt.transcript.text** ("video-id" — "s") grabs the data into a single string.
- **ytt.text** (seq-# — "s") converts a timed text into a single string.

Example 6: Using YTT

```
require arnold/common
require arnold/ytt

: main ( -- )
    "mScpHTIi-kM" ytt.transcript.timed
    .list
;
```

- This example prints out the timed text of the video with Video ID `mScpHTIi-kM`

DEMO:
Using the YTT library

Homework Step 1: Writing the Database microservice

- Write a microservice database server backed by a phash that acts like a remote hash, ie, it has 5 words `db#!`, `db#@`, `db#drop`, `db#contains?` and `db#size`
- Ensure that the dbase ONLY stores & fetches String values.
- Eg: `db#! ("key" "value" "dbase-name" —)`
- **IMPORTANT:** Use HTTP, not Smojo's binary protocol (ie, `dispatch http-server` not `bin-dispatch server`)
- Write an **asynchronous** stub that serializes/deserializes objects prior to inserting/ fetching them into/from the dbase.
- The async stubs should have the format: `xyz (* callback — async)`, where * is the inputs. The callback should receive the input `err true | value false`.
- Hint: `PACK BASE64> (object — "s")` will serialize any object. Similarly, `>BASE64 UNPACK ("s" — object)` will deserialize any object. These words are found in `*/java`
- Write a second **synchronous** stub to read/write data synchronously using HTTP-GET. Use the same serialization/deserialization words in this stub.
- The sync stub should have the format `xyz (* — err true | value false)`

Homework Step 2: Complete the YTT Download worker

- Complete the YTT download worker.
- Use an adaptive polling worker.
- The worker should:
 1. Read the video id from the submitted queue of the MQ server,
 2. Download the timed transcript and
 3. Save it as a key-value pair into the dbase named “transcripts”, using the **synchronous** stub
- Be sure to handle the case where a transcript does not exist. How is this indicated in the YTT library?
- Finally, update your deployment script to add all additional microservices. It should “just work”