Nutch 1.X RESTAPI

Nutch 1.x REST API v1.0

Introduction

This page documents the Nutch 1.X REST API v1.0.

It provides details on the type of REST calls which can be made to the Nutch 1.x REST API. Many of the API points are adapted from the ones provided by the Nutch 2.x REST API. One of the reasons to come up with a REST API is to integrate D3 to show visualizations about the working of a Nutch crawl.

Instructions to start Nutch Server

Follow the steps below to start an instance of the Nutch Server on localhost.

Starting Nutch Server

```
% cd runtime/local
% ./bin/nutch startserver -port <port_number> \[If the port option is not mentioned then by default the server
starts on port 8081\]
```

The different API calls that can be made are listed below.

REST API Calls

Administration

This API point is created in order to get server status and manage server's state.

Get server status

```
GET /admin
```

Response contains server startup date, availible configuration names, job history and currently running jobs.

```
{
   "startDate":1424572500000,
   "configuration":[
      "default"
   ],
   "jobs":[

   ],
   "runningJobs":[

   ]
}
```

Stop server

It is possible to stop running server using */admin/stop*.

```
GET /admin/stop
```

Response

```
Stopping in server on port 8081
```

Configuration

Configuration's list

```
GET /config
```

Response contains names of available configurations.

```
["default","custom-config"]
```

Configuration parameters

```
GET /config/{configuration name}

Examples:
GET /config/default
GET /config/custom-config
```

Response contains parameters with values

```
  {
   "anchorIndexingFilter.deduplicate":"false",
   "crawl.gen.delay":"604800000",
   "db.fetch.interval.default":"2592000",
   "db.fetch.interval.max":"7776000",
   ....
   ....
   }
```

Create configuration

Creates new Nutch configuration with given parameters.

```
POST /config/create

Examples:
POST /config/create
   {
      "configId":"new-config",
      "params":{"anchorIndexingFilter.deduplicate":"false",... }
   }

# curl
curl -X POST -H "Content-Type: application/json" http://localhost:8081/config/create -d '{"configId":"new-
config", "params":{"anchorIndexingFilter.deduplicate":"false"}}'
```

Response is created config's id.

```
    new-config
```

Get property value

```
GET /config/{configuration name}/{property}

Examples:
GET /config/default/anchorIndexingFilter.deduplicate
```

Response contains parameter's value as string

```
    false
```

Set property value

{

```
PUT /config/{configuration name}/{property}

Examples:
PUT /config/default/http.agent.name
```

Response contains parameter's value as string

```
    NUTCH_SOLR
```

Delete configuration

```
DELETE /config/{configuration name}

Examples:
DELETE /config/new-config
```

Jobs

This point allows job management, including creation, job information and killing of a job. For a complete tutorial, please follow How to run Jobs using the REST service.

Listing all jobs

```
curl -X GET -H 'Content-Type: application/json' -i http://localhost:8081/job
```

Response contains list of all jobs (running and history)

```
[
   {
      "id":"job-id-5977",
      "type":"FETCH",
      "confId":"default",
      "args":null,
      "result":null,
      "state":"FINISHED",
      "msg":"",
      "crawlId":"crawl-01"
   }
   {
      "id":"job-id-5978",
      "type":"PARSE",
      "confId":"default",
      "args":null,
      "result":null,
      "state":"RUNNING",
      "msg":"",
      "crawlId":"crawl-01"
   }
]
```

Get job info

```
GET /job/job-id-5977
```

Response

```
   {
      "id":"job-id-5977",
      "type":"FETCH",
      "confId":"default",
      "args":null,
      "result":null,
      "state":"FINISHED",
      "msg":"",
      "crawlId":"crawl01"
   }
```

Stop job

```
POST /job/job-id-5977/stop
```

Response

```
   true
```

Kill job

```
GET /job/job-id-5977/abort
```

}

Response
{

```
   true
```

}

Create job

Create job with given parameters. You should either specify Job Type(like INJECT, GENERATE, FETCH, PARSE, etc) or jobClassName.

```
curl -X POST -H 'Content-Type: application/json' -i http://localhost:8081/job/create --data '{"crawlId":"
crawl01", "type":"INJECT", "confId":"default", "args": {"url_dir":"seedFiles/seed-1641959745623", "crawldb":
"crawldb"}}'
```

Response object is provided below

```
{
  "id": "crawl01-default-INJECT-1877363907",
  "type": "INJECT",
  "confId": "default",
  "args": {
    "url_dir": "seedFiles/seed-1641959745623",
    "crawldb": "crawldb"
  },
  "result": null,
  "state": "RUNNING",
  "msg": "OK",
  "crawlId": "crawl01"
}
```

Seed Lists

Create seed list

The **/seed/create** endpoint enables the user to create a seedlist and return the temporary path of the file created. This path should be passed to the ***url_dir*** parameter of the **INJECT** job. It's also worth noting that the seed

```
curl -X POST -H 'Content-Type: application/json' -i http://localhost:8081/seed/create --data '{"name":"test","
seedUrls":["https://nutch.apache.org"]}'
```

Response is the relative file directory path. Note, this is relative to where the Nutch server was started. It's also worth noting that any seed lists which are created are persistent. That is to say they remain on disk even when nutch server is not running.

```
seedFiles/seed-1641959745623
```

Get seed lists

The **/seed** endpoint facilitates retrieval of any seedlists which were created during the current server runtime.

As of Nutch 1.18 seed lists generated by previous server runtime sessions will not be available if the server is shutdown and restarted.

Database

This point provides access to information stored in the CrawlDb.

```
POST /db/crawldb with following
{     "type":"stats",
      "confId":"default",
      "crawlId":"crawl01",
      "args":{"someParam":"someValue"}
}
```

The different values for the type parameter are - dump, topN and url. Their corresponding arguments can be found here.

Response contains information from the CrawlDbReader.java class. For the above mentioned request, the JSON response would like like-

```
{
    "retry 0":"8350",
    "minScore":"0.0",
    "retry 1":"96",
    "status":{
              "3":{"count":"21","statusValue":"db_gone"},
              "2":{"count":"594","statusValue":"db_fetched"},
              "1":{"count":"7721","statusValue":"db_unfetched"},
              "5":{"count":"86","statusValue":"db_redir_perm"},
              "4":{"count":"24","statusValue":"db_redir_temp"}
              },
    "totalUrls":"8446",
    "maxScore":"0.528",
    "avgScore":"0.029593771"
}
```


Note: If any other type than stats (like dump, topN, url) is used then the response will be a file (application-octet-stream).

More

Description of more API points coming soon.