NutchRESTAPI

Nutch 2.x REST API

Introduction

This page both documents and provides a UML graphic for the Nutch 2.X REST API.

It explains the logic behind the entire API and also provides detail on the type of REST calls which can be made to the Nutch 2.x REST API. This can be read in conjunction with the documentation on bin/nutch nutchserver command.

REST API Calls

Administration

Responsible class is *AdminResource*. 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":1403716000012,
   "configuration":[
      "default"
   ],
   "jobs":[

   ],
   "runningJobs":[

   ]
}
```

}

Stop server

It is possible to stop running server using */admin/stop*. You can use non-mandatory *force* parameter, if you want to stop server despite running tasks.
{

```
GET /admin/stop
GET /admin/stop?force=true
```

}

Response
{

```
Stopping in 5 seconds.
```

}

Jobs

Responsible class is *JobResource*. This point is created for job's management.

Listing jobs

{

```
GET /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":"crawl-01"
   }
```

}

Stop job

{

```
GET /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 JobType or jobClassName.
{

```
POST /job/create
   {
      "crawlId":"crawl-01",
      "type":"FETCH",
      "confId":"default",
      "args":{"someParam":"someValue"}
   }

POST /job/create
   {
      "crawlId":"crawl-01",
      "jobClassName":"org.apache.nutch.fetcher.FetcherJob"
      "confId":"default",
      "args":{"someParam":"someValue"}
   }
```

}

Response is created job's id.
{

```
    job-id-43243
```

}

Configuration

Configuration's list

{

```
GET /config
```

}

Response contains names of availible 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",
   "db.fetch.retry.max":"3",
   ....
   ....
   }
```

}

Get property value

{

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

Examples:
GET /config/default/db.fetch.retry.max
GET /config/custom-config/crawl.gen.delay
```

}

Response contains parameter's value as string
{

```
    604800000
```

}

Create configuration

Creates new nutch configuration with given parameters. It force field is true, then already existing configuration will be overrided, otherwise not.
{

```
POST /config/{configuration name}

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

}

Response is created config's id.
{

```
    new-config
```

}

Delete configuration

{

```
DELETE /config/{configuration name}

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

}

Update property value

{

```
PUT /config/{property name}/
value={value}

Examples:
PUT /config/anchorIndexingFilter.deduplicate
value=true
```

}

Database

Responsible class is *DbResource*. This point is created in order to get data from database.

Run query

Examples:
{

```
POST /db
   {

   }

POST /db
   {
      "fields": ["headers"]
   }

POST /db
   {
      "batchId": "batch-id"
   }

POST /db
   {
      "startKey":"http://google.com",
      "endKey":"http://yahoo.com",
      "isKeysReversed":"false",
   }


POST /db
   {
      "startKey":"com.google",
      "endKey":"com.yahoo",
      "isKeysReversed":"true"
   }
```

}
Response contains data from database with filtered fields.
{

```
    {
   "values":[
      {
         "headers":{

         },
         "status":0,
         "markers":{

         },
         "modifiedTime":0,
         "score":0.0,
         "prevModifiedTime":0,
         "url":"http://google.com",
         "__g__dirty":"\\x00\\x00\\x00\\x00",
         "fetchInterval":0,
         "prevFetchTime":0,
         "inlinks":{

         },
         "retriesSinceFetch":0,
         "outlinks":{

         },
         "fetchTime":0,
         "metadata":{

         }
      }
   ]
}
```


}

Rest API improvement proposals

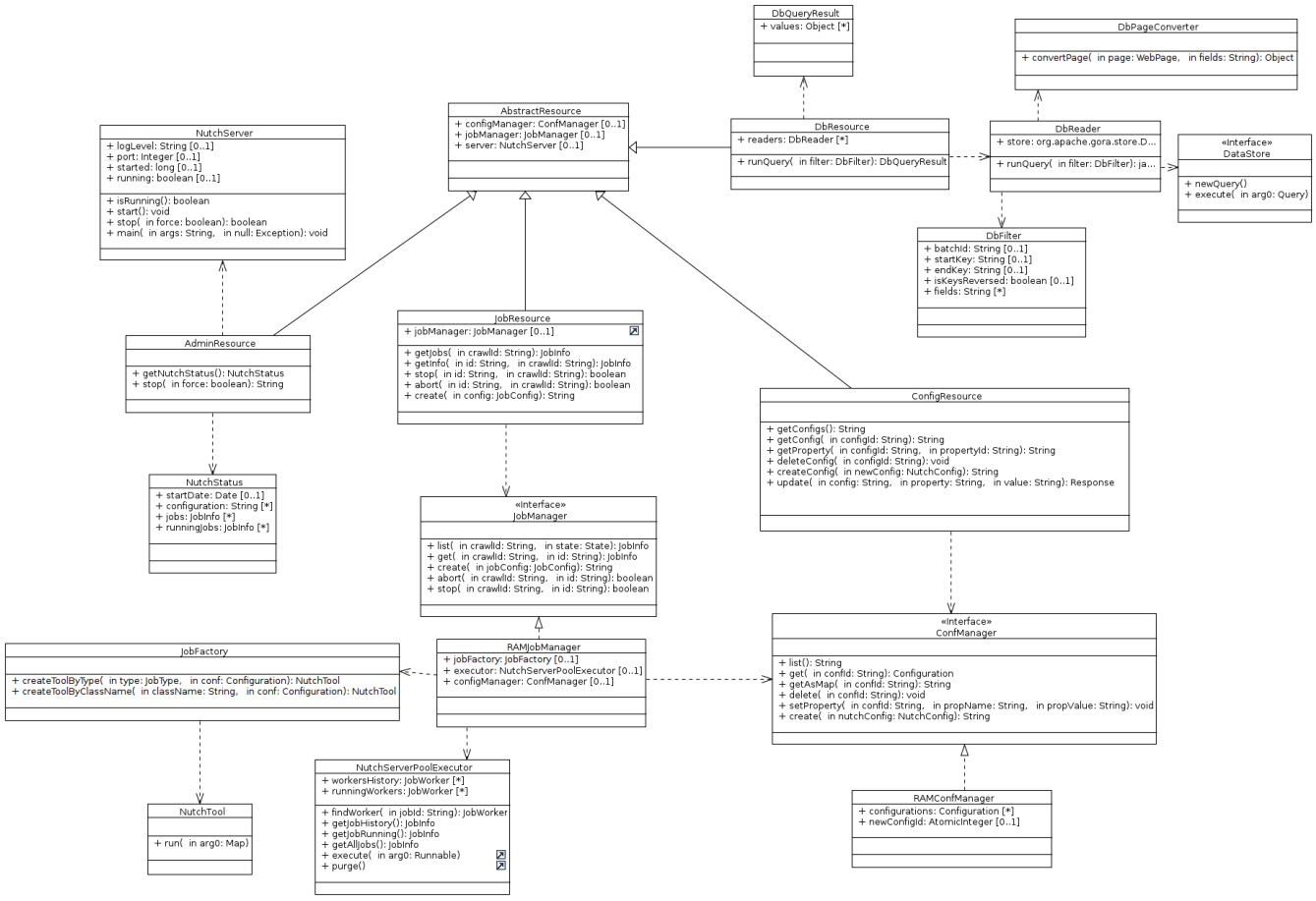
- Naming of RAMConfManager and RAMJobManager
- crawlId support in DbFilter
- remove jobs from status, add jobHistory
- Check, if POST method is situable for /db requests.

Nutch Jobs

UML Graphic

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

The graphic below displays the REST API architecture and described the classes as well as the role and context within the API operation.



Some comments about class roles in Nutch API.

- NutchServer - entry point. Parses commandline parameters and configures Restlet application through JAX-RS API.
- AbstractResource - abstract JAX-RS resource. Other JAX-RS extend it in order to get references to ConfManager, JobManager and NutchServer.
- JobFactory - factory class, which creates job objects based on JobType or class name.
- DbReader - manages connections to web store, processes filter and runs Gora query.
- DbIterator - navigates through selected data, skips non-relevant records
- DbPageConverter - converts database record into Nutch API model object
- NutchServerPoolExecutor - manages running jobs and job's history.
- RAMConfManager - manages nutch configuration in memory
- RAMJobManager - stores job info in memory, job execution

Related Documentation

- bin/nutch nutchserver - run a (local) Nutch server on a user defined port.

back to FrontPage