
JSONParse

Release 0.1.0

Luke Du

Jun 17, 2021

CONTENTS:

1	Python Code	1
1.1	JSON Parse	1
2	UnitTest Code	7
2.1	test-jsonparse	7
3	Indices and tables	9
	Python Module Index	11
	Index	13

PYTHON CODE

1.1 JSON Parse

1.1.1 The JSON solution

- **File name:** json_utils.py
 - **Arthor:** Luke Du
 - **Updates:**
 - 17JUN2021-LD load multiple JSON data. Instead of just load one data, change code to append way
- Parse JSON data directly using JSON format mapping.

JsonUtils CLASS

```
class jsonutils.JsonUtils (csv_delim=',', json_txn_id_name='txn_uuid', table_name_prefix="",  
                           flag_json_array='__JSON_array__')
```

Bases: object

variable member initialization in `__init__` function

- **csv_delim:**
 - specific symbol for csv format
 - The symbol in json data need to removed/replaced before parsing
 - Used for special symbol when import data into database
- **json_txn_id_name:**
 - specific column name for json data
 - One json record may be parsed to several tables
 - This column identify which record it is
 - It will be used to link tables together
- **table_name_prefix:** Optional, specify table name with this prefix
- **flag_json_array:**
 - Special string
 - When go through json data, indicate the tag is json list
 - Later this tag will becomes separate table

- **json_data**: data loaded in json format using json module
- **pathlist**: all path in data, columns in table later
- **arraylist**: all array in data, tables later
- **map**:
 - JSON format map
 - can be generated by `self.table_plan_json`
 - can be exported to JSON format by `self.json_map_export`
 - can be loaded from JSON file by `self.json_map_import`
 - can be exported to CSV format for customization by `self.map_export_csv`
 - can be loaded from CSV file with customization by `self.map_import_csv`
- **parsed_tables**: parsed cvs tables
- **map_path**: path list from map
- **map_array**: array list from map

add_new_path_to_map (*new_path_list*)
Add new paths to map

- Assume no new table need to be added. If there exist new tables, will work on it in future.
- based on *new_path_list* from the new JSON data and **map_path**
- Add new path into **map**

append_from_list (*jsonlist=None*)
Append to JSON data from JSON list

- use list extend method

compute_all_paths ()
Compute all paths in JSON data

- call **get_paths** out of this class (see below), work on **json_data**
- Store output into **arraylist** (table) and **pathlist**

get_json_len ()

- output length of JSON data

json_map_export (*map_file=None*)
export map as JSON format

- export map into *map_file*
- later the JSON map file can be used to load JSON data
- user can modify (not recommended. We recommend to load JSON map; export to csv file; edit csv file; load from csv file, and export modified map in JSON format)

json_map_import (*map_file=None*)
import map from JSON format

load_from_file (*df=None*)
load JSON data from file

- Valid JSON data can be JSON list(*[]*). The JSON lines (one line per JSON transaction) may not work.

- JSON data stored into **json_data**

load_from_list (*jsonlist=None*)
load JSON data from data list

- Convert *jsonlist* to *jstr* then call **load_from_string**

load_from_string (*jstr=None*)
load JSON data from string

- Valid JSON data can be JSON list(*[]*). The JSON lines (one line per JSON transaction) may not work.
- JSON data stored into **json_data**

map_export_csv (*map_csv=None*)
export map as csv format

- user can modify and import again

map_import_csv (*map_csv=None*)
import map from csv format

- the tool to change JSON map

map_to_allpath ()
Compute all path from map

- This method is used when we like to check/update map using new JSON data
- When we import map (from JSON or CSV format) into **map**
- this method analyze **map** and store all path and array (table) into **map_path** and **map_array**
- Later we can check new JSON data with **map_path** for new paths.

parse_to_csv ()
parse JSON data to csv format using map

- Based on data in **json_data** and map in **map**, parse JSON data
- Store CSV format data into **parsed_tables**

postgres_ddl (*sql_file=None, schema_name='default_schema'*)
generate postgresql queries of table DDL based on map

- Based on **map**, create table DDL using *schema_name*
- Store SQL query into *sql_file*

table_plan_json ()
create map in json format

- Based on **arraylist** and **pathlist**, compute map in Python dictionary
- Store map (python dictionary) into **map**

jsonutils.compute_table_content (*json_data, seq_list, json_txn_id_name='txn_uuid'*)
compute table content based on seq_list (with array path)

- Based on *seq_list*, go to the level where data stored in.
- for each level, keep *seq_no* to the lower level.
- At the lowest level, combine all json element into js array.
- return js array.
- called by **parse_to_csv**

`jsonutils.get_paths(source, flag_json_array)`
get full path

- This is out of CLASS **JsonUtils**
- Code originally from <https://stackoverflow.com/questions/51488240/python-get-json-keys-as-full-path>
- This is recursive function: call itself
- called by **compute_all_paths**

`jsonutils.name_from_path(path, name_list)`
Get variable name from path

- Giving *path* and existing *name_list*
- compute name for this *path* (last element of the path)
- If exist in *name_list*, create random one for later to rename
- called by **table_plan_json** and **add_new_path_to_map**

`jsonutils.parse(json_data, column_list, seq_list=None, debug_idx=0, json_txn_id_name='txn_uuid', csv_delim=',')`
one level parse

- parse *json_data* based on *column_list*
- Store the parsed result as text
- Combine parsed text result using *csv_delim*
- With *seq_list*, combine *seq_no* value to result
- called by **parse_to_csv**

`jsonutils.parse_tags_wo_arr(json_data, tags)`
parse multiple level of tag without array

In JSON format, the data can be stored in multiple levels. If all these levels are not array (list of values, which means, one transaction have multiple value for that tag), we can still treat it as transaction level variable (no need to store into separate table).

- Given *json_data* and *tags*
- for each level of *tags*, go inside of JSON element in *json_data*
- If value does not exist in *tags* path, just return *None*
- else, return the value according to *tags* path.
- called by **parse** and **compute_table_content** (out of class **JsonUtils**)

`jsonutils.table_seq_list(path, arraylist)`
compute seqList based on table path in position of arraylist

- Given one *path*, compute which table it should be, and which *seq_no* it should have.
- When *path* in *arraylist* (table) is subset of *path*, this *path* belong to this table. If this table have *seq_no*, this *seq_no* need to be used.
- Set table name (last tag in table path) into column name
- the *seq_list* will show which level of the table it is, like header table is level 0; item table is level 1; and itemdiscount table is level 2; etc.
- if the table is level 0, no *seq_list*;
- if the table is level 1, *seq_list* has one element;

- if the table is level 2, seq_list have two elements.
- called by **table_plan_json**

UNITTEST CODE

2.1 test-jsonparse

2.1.1 Demo How to Use JSONparse

- **Program file:** test_jsonparse.py
- **Client :** demo using sample JSON data file
- **Updates :**

The purpose of this file is to demonstrate importing data files to database.

Run this test under upper folder of *tests*

```
python -B -m unittest tests.test_jsonparse.test_map_gen
```

or

```
python -B -m unittest tests.test_jsonparse.test_parse
```

etc.

- The dbinterface.postgresql is used to connect to database
- You need to install it in your virtual environment

Common tests

- **test_map_gen** will do the following:
 - load into **json_data**
 - Compute all paths in data
 - Generate map (table_plan_json)
 - export to JSON format map file
 - generate postgres DDL SQL query to file
- **test_map2csv** will do the following:
 - Import JSON format map file
 - Export to CSV format map file
 - Then user can modify CSV format map file as needed
- **test_csv2map** will do the following:

- Import CSV format map file
- generate postgres DDL SQL query to file (based on user changedd map)
- Export to JSON format map file
- Later user can parse data using this new map and import to database
- **test_parse** will do the following:
 - Decrypt data file
 - load into **json_data**
 - Import JSON format map file
 - Parse JSON data based on map
 - Import parsed data into database (assume tables in database has been created using DDL)

The python functions

```
test_jsonparse.GetPostgreSQLLoginInfo()  
    • Get database login information from pem file  
class test_jsonparse.TestJSONparse (methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_csv2map()  
        test function: import CSV format map  
  
    test_json_append()  
        test if we can append json in Python * test driven development example * test and use in production  
  
    test_json_gen_2()  
        test if we can append json in Python * prepare data for test_json_append function below  
  
    test_json_utils_append()  
        test if we can append json in Python * test driven development example * test and use in production  
  
    test_map2csv()  
        test function: import map to CSV file for user revising  
  
    test_map_gen()  
        test function: generate map based on data  
  
    test_parse()  
        test function: parse JSON data based on map; import into database  
  
test_jsonparse.flush_to_db (ju=None, db_conn=None, schema=None, truncate_before_flush=True)  
    flush data from memory to greenplum tables  
  
test_jsonparse.get_db_conn()  
    define database connection using Sql under dbinterface
```

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

j

`jsonutils`, 1

t

`test_jsonparse`, 7

A

`add_new_path_to_map()` (*jsonutils.JsonUtils method*), 2
`append_from_list()` (*jsonutils.JsonUtils method*), 2

C

`compute_all_paths()` (*jsonutils.JsonUtils method*), 2
`compute_table_content()` (*in module jsonutils*), 3

F

`flush_to_db()` (*in module test_jsonparse*), 8

G

`get_db_conn()` (*in module test_jsonparse*), 8
`get_json_len()` (*jsonutils.JsonUtils method*), 2
`get_paths()` (*in module jsonutils*), 3
`GetPostgreSQLLoginInfo()` (*in module test_jsonparse*), 8

J

`json_map_export()` (*jsonutils.JsonUtils method*), 2
`json_map_import()` (*jsonutils.JsonUtils method*), 2
`jsonutils`
 module, 1
`JsonUtils` (*class in jsonutils*), 1

L

`load_from_file()` (*jsonutils.JsonUtils method*), 2
`load_from_list()` (*jsonutils.JsonUtils method*), 3
`load_from_string()` (*jsonutils.JsonUtils method*), 3

M

`map_export_csv()` (*jsonutils.JsonUtils method*), 3
`map_import_csv()` (*jsonutils.JsonUtils method*), 3
`map_to_allpath()` (*jsonutils.JsonUtils method*), 3
module
 jsonutils, 1
 test_jsonparse, 7

N

`name_from_path()` (*in module jsonutils*), 4

P

`parse()` (*in module jsonutils*), 4
`parse_tags_wo_arr()` (*in module jsonutils*), 4
`parse_to_csv()` (*jsonutils.JsonUtils method*), 3
`postgres_ddl()` (*jsonutils.JsonUtils method*), 3

T

`table_plan_json()` (*jsonutils.JsonUtils method*), 3
`table_seq_list()` (*in module jsonutils*), 4
`test_csv2map()` (*test_jsonparse.TestJSONparse method*), 8
`test_json_append()`
 (*test_jsonparse.TestJSONparse method*), 8
`test_json_gen_2()` (*test_jsonparse.TestJSONparse method*), 8
`test_json_utils_append()`
 (*test_jsonparse.TestJSONparse method*), 8
`test_jsonparse`
 module, 7
`test_map2csv()` (*test_jsonparse.TestJSONparse method*), 8
`test_map_gen()` (*test_jsonparse.TestJSONparse method*), 8
`test_parse()` (*test_jsonparse.TestJSONparse method*), 8
`TestJSONparse` (*class in test_jsonparse*), 8