pyquery-ql Documentation

Release 0.1

Carol Willing

Dec 15, 2017

Contents:

Query GitHub API v4 using pyquery-ql a lightweight GraphQL client.

CHAPTER 1

Introduction

1.1 What is pyquery-ql?

It's a repository for examples and tips when querying GraphQL APIs using *Python* and the popular *Jupyter notebook*. It provides examples on using the elegant *requests* library and *pandas* to do data collection and interactive data exploration.

pyquery-ql is also a very early prototype for a lightweight Python GraphQL client. It's based on Python 3.6 or higher and takes advantage of f-strings (if you haven't tried them yet, please do).

1.2 Why create a lightweight GraphQL client for Python?

In data science and scientific programming, we do lots of queries for data from different sources. With the increased use of GraphQL, it's helpful to have a lightweight client to do quick queries.

With Jupyter and nteract notebooks, it's critical to one's workflow to be able to prototype and iterate from within the notebook. While full featured clients, like Apollo Client, and explorers, such as GitHub's GraphQL Explorer, are wonderful, the user must leave the notebook to see their benefits.

Using the Jupyter or nteract notebooks, a user explores data, and GraphQL is fun to use since it's very easy to introspect the data source. A lightweight Python client provides a simple way to bridge the data and the Python scientific stack, especially pandas, from within the interactive notebook.

CHAPTER 2

Getting Started

2.1 Installation

Supports Python 3.6+

1. Clone the repo.

```
git clone https://github.com/willingc/pyquery-ql.git
```

2. Set up your GitHub API token to avoid rate limit and export to your environment:

```
export GITHUB_API_TOKEN = '......'
```

2.1.1 pip installation instructions

1. From the root of the cloned repo:

```
pip install -r requirements.txt
```

2.1.2 conda installation instructions

1. Create the conda environment:

```
conda env create -f environment.yml
```

2. Activate the environment:

```
source activate pyquery
```

2.2 Usage

To run:

```
python3 pyquery-ql.py
```

CHAPTER 3

Notes about GraphQL

3.1 GraphQL basics

- GraphQL
  - describe your data
  - ask for what you want
  - get predictable results
- A GraphQL service defines **types** and **fields** on those types. A function is created for each field on each type.
  - **field** can be a specific type (like String) or an Object
  - **arguments** can be passed to fields when making a query

```
{
  human(id: "1000") {
    name
    height
  }
}
```

- GraphQL schema and types

```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

  - GraphQL Object type `Character`
  - fields `name` and `appearsIn`
  - Scalar type `String`
  - `!` non-nullable (will always return a value when queried)

  - array `[Episode]`
- Arguments
  - every field can have 0 or more arguments
  - all arguments are named
  - all arguments are passed by name specifically (unlike JS or Python where functions take a list of ordered arguments
  - required or optional
- Scalar types
  - fields that resolve to real data
  - leaves of the query
  - default scalar types:
    * `Int`
    * `Float`
    * `String`
    * `Boolean`
    * `ID` unique identifier
  - custom scalar types are possible i.e. `Date`
  - `Enums`
- Modifiers: Non-Null and List (array)
  - List [] can be null
  - a member of the list can not be null or it will generate an error
  - []! the actual list can not be null though a member can be null
- **Introspection** is a beautiful thing; ask for `__schema` for all the good stuff on the types

3.2 GraphQL Client needs

- Pagination
  - plurals
  - slicing
  - pagination and edges
  - Connections
- Caching
  - ID
  - pros and cons of globally unique IDs
  - server defined or client derived are both options

3.3 GraphQL spec

CHAPTER 4

License

MIT

CHAPTER 5

Indices and tables

- genindex
- modindex
- search