ArmindoFlores / ao3_api Public

Code Issues 14 Pull requests 6 Actions Projects Security Insights

master Go to file Code

ArmindoFlores Merge pull request #80 from williln/add-recomme... ... on Oct 26, 2022 280

AO3	Merge pull request #80 from willlln/add-recommend...	5 months ago
dist	Changed version numbers and added new binaries	10 months ago
docs	Reverted use of 'py3' code tags	2 years ago
LICENSE	Add files via upload	5 years ago
README.md	Minor edits for spelling/grammar	2 years ago
index.rst	Moved mkdocs.yml and index.rst	3 years ago
mkdocs.yml	Added more documentation	3 years ago
requirements.txt	Added requirements.txt	2 years ago
setup.cfg	Add files via upload	5 years ago
setup.py	Changed version numbers and added new binaries	10 months ago

docs passing

AO3 API

This is an unofficial API that lets you access some of AO3's (archiveofourown.org) data through Python.

Installation

Use the package manager pip to install AO3 API.

```
pip install ao3_api
```

Github

https://github.com/ArmindoFlores/ao3_api

Usage

This package is divided in 9 core modules: works, chapters, users, series, search, session, comments, extra, and utils.

Works

One of the most basic things you might want to do with this package is loading a work and checking its statistics and information. To do that, you'll need the `AO3.Work` class.

We start by finding the *workid* of the work we want to load. We do that either by using `AO3.utils.workid_from_url(url)` or by just looking at the url ourselves. Let's take a look:

About

An unofficial archiveofourown.org (AO3) API for python

Readme

MIT license

117 stars

9 watching

40 forks

Releases 9

AO3 API v2.3.0 Latest
on May 31, 2022

+ 8 releases

Packages

No packages published

Used by 25

+ 17

Contributors 17

+ 6 contributors

Languages

● Python 100.0%

```
import AO3

url = "https://archiveofourown.org/works/14392692/chapters/33236241"
workid = AO3.utils.workid_from_url(url)
print(f"Work ID: {workid}")
work = AO3.Work(workid)
print(f"Chapters: {work.nchapters}")
```

After running this snippet, we get the output:

```
Work ID: 14392692
Chapters: 46
```

It's important to note that some works may not be accessible to guest users, and in this case you will get 0 chapters as an output, and the error `AO3.utils.AuthError: This work is only available to registered users of the Archive` if you try to load it. Nontheless, we can still do a lot more with this Work object: Lets try to get the first 20 words of the second chapter.

```
import AO3

work = AO3.Work(14392692)

print(work.chapters[1].title)  # Second chapter name
text = work.chapters[1].text  # Second chapter text
print(' '.join(text.split(" ")[:20]))
```

```
What Branches Grow Meaning
December 27, 2018

Christmas sucked this year, and Shouto’s got the black eye to prove it.
Things had started out well enough,
```

The objects in work.chapters are of type `AO3.Chapter` . They have a lot of the same properties as a `Work` object would.

Another thing you can do with the work object is download the entire work as a pdf or e-book. At the moment you can download works as AZW3, EPUB, HTML, MOBI, and PDF files.

```
import AO3

work = AO3.Work(14392692)

with open(f"{work.title}.pdf", "wb") as file:
    file.write(work.download("PDF"))
```

Advanced functionality

Usually, when you call the constructor for the `Work` class, all info about it is loaded in the `__init__()` function. However, this process takes quite some time (~1-1.5 seconds) and if you want to load a list of works from a series, for example, you might be waiting for upwards of 30 seconds. To avoid this problem, the `Work.reload()` function, called on initialization, is a "threadable" function, which means that if you call it with the argument `threaded=True` , it will return a `Thread` object and work in parallel, meaning you can load multiple works at the same time. Let's take a look at an implementation:

```
import AO3
import time

series = AO3.Series(1295090)

works = []
threads = []
start = time.time()
```

```
for work in series.work_list:
    works.append(work)
    threads.append(work.reload(threaded=True))
for thread in threads:
    thread.join()
print(f"Loaded {len(works)} works in {round(time.time()-start, 1)} seconds.")
```

```
Loaded 29 works in 2.2 seconds.
```

The `load=False` inside the `Work` constructor makes sure we don't load the work as soon as we create an instance of the class. In the end, we iterate over every thread and wait for the last one to finish using `.join()`. Let's compare this method with the standard way of loading AO3 works:

```
import AO3
import time

series = AO3.Series(1295090)

works = []
start = time.time()
for work in series.work_list:
    work.reload()
    works.append(work)

print(f"Loaded {len(works)} works in {round(time.time()-start, 1)} seconds.")
```

```
Loaded 29 works in 21.6 seconds.
```

As we can see, there is a significant performance increase. There are other functions in this package which have this functionality. To see if a function is "threadable", either use `hasattr(function, "_threadable")` or check its `__doc__` string.

To save even more time, if you're only interested in metadata, you can load a work with the `load_chapters` option set to False. Also, be aware that some functions (like `Series.work_list` or `Search.results`) might return semi-loaded `Work` objects. This means that no requests have been made to load this work (so you don't have access to chapter text, notes, etc...) but almost all of its metadata will already have been cached, and you might not need to call `Work.reload()` at all.

The last important information about the `Work` class is that most of its properties (like the number of bookmarks, kudos, the authors' names, etc...) are cached properties. That means that once you check them once, the value is stored and it won't ever change, even if those values change. To update these values, you will need to call `Work.reload()`. See the example below:

```
import AO3

sess = AO3.GuestSession()
work = AO3.Work(16721367, sess)
print(work.kudos)
work.leave_kudos()
work.reload()
print(work.kudos)
```

```
392
393
```

Users

Another useful thing you might want to do is get information on who wrote which works / comments. For that, we use the `AO3.User` class.

```
import AO3
```

```
user = AO3.User("bothersomepotato")
print(user.url)
print(user.bio)
print(user.works)  # Number of works published
```

```
https://archiveofourown.org/users/bothersomepotato
University student, opening documents to write essays but writing this stuff
instead. No regrets though. My Tumblr, come chat with -or yell at- me if you
feel like it! :)
2
```

README.md

Search

To search for works, you can either use the `AO3.search()` function and parse the BeautifulSoup object returned yourself, or use the `AO3.Search` class to automatically do that for you.

```
import AO3
search = AO3.Search(any_field="Clarke Lexa", word_count=AO3.utils.Constraint(50
search.update()
print(search.total_results)
for result in search.results:
  print(result)
```

```
3074
<Work [five times lexa falls for clarke]>
<Work [an incomplete list of reasons (why Clarke loves Lexa)]>
<Work [five times clarke and lexa aren’t sure if they're a couple or not]>
<Work [Chemistry]>
<Work [The New Commander (Lexa Joining Camp Jaha)]>
<Work [Ode to Clarke]>
<Work [it's always been (right in front of me)]>
<Work [The Girlfriend Tag]>
<Work [The After-Heda Chronicles]>
<Work [The Counter]>
<Work [May We Meet Again]>
<Work [No Filter]>
<Work [The Games We Play]>
<Work [A l'épreuve des balles]>
<Work [Celebration]>
<Work [Another level of fucked up]>
<Work [(Don't Ever Want to Tame) This Wild Heart]>
<Work [Self Control]>
<Work [Winter]>
<Work [My only wish]>
```

You can then use the workid to load one of the works you searched for. To get more then the first 20 works, change the page number using

```
search.page = 2
```

Session

A lot of actions you might want to take might require an AO3 account. If you already have one, you can access those actions using an AO3.Session object. You start by logging in using your username and password, and then you can use that object to access restricted content.

```
import AO3

session = AO3.Session("username", "password")
print(f"Bookmarks: {session.bookmarks}")
session.refresh_auth_token()
print(session.kudos(AO3.Work(18001499, load=False))
Bookmarks: 67
True
```

We successfully left kudos in a work and checked our bookmarks. The `session.refresh_auth_token()` is needed for some activities such as leaving kudos and comments. If it is expired or you forget to call this function, the error `AO3.utils.AuthError: Invalid authentication token. Try calling session.refresh_auth_token()` will be raised.

You can also comment / leave kudos in a work by calling `Work.leave_kudos()` / `Work.comment()` and provided you have instantiated that object with a session already (`AO3.Work(xxxxxx, session=sess)` or using `Work.set_session()`). This is probably the best way to do so because you will run into less authentication issues (as the work's authenticity token will be used instead).

If you would prefer to leave a comment or kudos anonymously, you can use an `AO3.GuestSession` in the same way you'd use a normal session, except you won't be able to check your bookmarks, subscriptions, etc. because you're not actually logged in.

Comments

To retrieve and process comment threads, you might want to look at the `Work.get_comments()` method. It returns all the comments in a specific chapter and their respective threads. You can then process them however you want. Let's take a look:

```
from time import time

import AO3


work = AO3.Work(24560008)
work.load_chapters()
start = time()
comments = work.get_comments(5)
print(f"Loaded {len(comments)} comment threads in {round(time()-start, 1)} seco
for comment in comments:
    print(f"Comment ID: {comment.id}\nReplies: {len(comment.get_thread())}")
```

```
Loaded 5 comment threads in 1.8 seconds

Comment ID: 312237184
Replies: 1
Comment ID: 312245032
Replies: 1
Comment ID: 312257098
Replies: 1
Comment ID: 312257860
Replies: 1
Comment ID: 312285673
Replies: 2
```

Loading comments takes a very long time so you should try and use it as little as possible. It also causes lots of requests to be sent to the AO3 servers, which might result in getting the error `utils.HTTPError: We are being rate-limited. Try again in a while or reduce the number of requests`. If that happens, you should try to space out your requests or reduce their number. There is also the option to enable request limiting using `AO3.utils.limit_requests()`, which make it so you can't make more than x requests in a certain time window. You can also reply to comments using the `Comment.reply()` function, or delete one (if it's yours) using `Comment.delete()`.

Extra

AO3.extra contains the the code to download some extra resources that are not core to the functionality of this package and don't change very often. One example would be the list of fandoms recognized by AO3. To download a resource, simply use `AO3.extra.download(resource_name)` . To download every resource, you can use `AO3.extra.download_all()` . To see the list of available resources, use `AO3.extra.get_resources()` .

Contact info

For information or bug reports please contact francisco.rodrigues0908@gmail.com.

License