aiohttp_session Documentation

Release 2.9.0-

Andrew Svetlov

Nov 15, 2019

Contents

The library provides sessions for aiohttp.web.

The current version is 2.9

CHAPTER 1

Usage

The library allows to store user-specific data into session object.

The session object has dict-like interface (operations like `session[key] = value` or `value = session[key]` etc. are supported).

Before processing session in web-handler you have to register *session middleware* in `aiohttp.web.Application`.

A trivial usage example:

```
import time
from aiohttp import web
from aiohttp_session import get_session, setup
from aiohttp_session.cookie_storage import EncryptedCookieStorage

async def handler(request):
    session = await get_session(request)
    session['last_visit'] = time.time()
    return web.Response(text='OK')

def init():
    app = web.Application()
    setup(app,
        EncryptedCookieStorage(b'Thirty  two  length  bytes  key.'))
    app.router.add_route('GET', '/', handler)
    return app

web.run_app(init())
```

All storages uses HTTP Cookie named `AIOHTTP_COOKIE_SESSION` for storing data.

Available session storages are:

- *`SimpleCookieStorage`* – keeps session data as plain JSON string in cookie body. Use the storage only for testing purposes, it's very non-secure.

- *EncryptedCookieStorage* – stores session data into cookies like *SimpleCookieStorage* does but encodes the data via *cryptography* Fernet cipher.

  For key generation use `cryptography.fernet.Fernet.generate_key()` method.

  Requires *cryptography* library:

```
$ pip3 install aiohttp_session[secure]
```

- *RedisStorage* – stores JSON-ed data into *redis*, keeping into cookie only redis key (random UUID).

  Inside redis the key will be saved as COOKIENAME_VALUEOFTHECOOKIE. For example if inside the browser the cookie is saved with name `'AIOHTTP_SESSION'` (default option) and value `e33b57c7ec6e425eb626610f811ab6ae` (a random UUID) they key inside redis will be `AIOHTTP_SESSION_e33b57c7ec6e425eb626610f811ab6ae`.

  Requires *aioredis* library:

```
$ pip install aiohttp_session[aioredis]
```

- `MemcachedStorage` – the same as Redis storage but uses Memcached database.

  Requires *aiomcache* library:

```
$ pip install aiohttp_session[aiomcache]
```

CHAPTER 2

Installation

```
$ pip3 install aiohttp_session
```

CHAPTER 3

Source code

The project is hosted on GitHub

Please feel free to file an issue on bug tracker if you have found a bug or have some suggestion for library improvement.

The library uses Travis for Continuous Integration.

CHAPTER 4

Dependencies

- Python 3.5.3+
- *cryptography* for `EncryptedCookieStorage`
- *aioredis* for `RedisStorage`.

CHAPTER 5

Third party extensions

- aiohttp_session_mongo

CHAPTER 6

License

aiohttp_session is offered under the Apache 2 license.

Contents:

6.1 Reference

6.1.1 Public functions

aiohttp_session.**get_session**(*request*)
A coroutine for getting session instance from request object.

See example below in *Session* section for *get_session()* usage.

aiohttp_session.**new_session**(*request*)
A coroutine for getting a new session regardless of whether a cookie exists.

> **Warning:** Always use *new_session()* instead of *get_session()* in your login views to guard against Session Fixation attacks!

Example usage:

```
from aiohttp_session import new_session

async def handler(request):
    session = await new_session(request)
    session.new == True # This will always be True
```

aiohttp_session.**session_middleware**(*storage*)
Session middleware factory.

Create session middleware to pass into aiohttp.web.Application constructor.

storage is a session storage instance (object used to store session data into cookies, Redis, database etc., class is derived from *AbstractStorage*).

See also:

Session storages

Note: *setup()* is new-fashion way for library setup.

---

aiohttp_session.**setup**(*app*, *storage*)
Setup session support for given *app*.

The function is shortcut for:

```
app.middlewares.append(session_middleware(storage))
```

app is aiohttp.web.Application instance.

storage is a session storage instance (object used to store session data into cookies, Redis, database etc., class is derived from *AbstractStorage*).

See also:

Session storages

6.1.2 Session

class aiohttp_session.**Session**
Client's session, a namespace that is valid for some period of continual activity that can be used to represent a user's interaction with a web application.

Warning: Never create *Session* instances by hands, retieve those by *get_session()* call.

The *Session* is a MutableMapping, thus it supports all dictionary methods, along with some extra attributes and methods:

```
from aiohttp_session import get_session

async def handler(request):
    session = await get_session(request)
    session['key1'] = 'value 1'
    assert 'key2' in session
    assert session['key2'] == 'value 2'
    # ...
```

created
Creation UNIX TIMESTAMP, the value returned by time.time() for very first access to the session object.

identity
Client's identity. It may be cookie name or database key. Read-only property. For change use *Session.set_new_identity()*.

new
A boolean. If new is True, this session is new. Otherwise, it has been constituted from data that was already serialized.

changed()
Call this when you mutate a mutable value in the session namespace. See the note below for details on when, and why you should call this.

Note: Keys and values of session data must be JSON serializable when using one of the included storage backends. This means, typically, that they are instances of basic types of objects, such as strings, lists, dictionaries, tuples, integers, etc. If you place an object in a session data key or value that is not JSON serializable, an error will be raised when the session is serialized.

If you place a mutable value (for example, a list or a dictionary) in a session object, and you subsequently mutate that value, you must call the changed() method of the session object. In this case, the session has no way to know that is was modified. However, when you modify a session object directly, such as setting a value (i.e., `__setitem__`), or removing a key (e.g., `del` or `pop`), the session will automatically know that it needs to re-serialize its data, thus calling *changed()* is unnecessary. There is no harm in calling *changed()* in either case, so when in doubt, call it after you've changed sessioning data.

---

invalidate()
Call this when you want to invalidate the session (dump all data, and – perhaps – set a clearing cookie).

set_new_identity(*identity*)
Call this when you want to change the *identity*.

> **Warning:** Never change *identity* of a session which is not new.

6.1.3 Session storages

aiohttp_session uses storages to save/load persistend session data.

Abstract Storage

All storages should be derived from *AbstractStorage* and implement both *load_session()* and *save_session()* methods.

class aiohttp_session.**AbstractStorage**(*cookie_name="AIOHTTP_SESSION"*, ***, *domain=None*, *max_age=None*, *path='/'*, *secure=None*, *httponly=True*, *encoder=json.dumps*, *decoder=json.loads*)
Base class for session storage implementations.

It uses HTTP cookie for storing at least the key for session data, but some implementations may save all session info into cookies.

cookie_name – name of cookie used for saving session data.

domain – cookie's domain, `str` or `None`.

max_age – cookie's max age, `int` or `None`.

path – cookie's path, `str` or `None`.

secure – cookie's secure flag, `bool` or `None` (the same as `False`).

httponly – cookie's http-only flag, `bool` or `None` (the same as `False`).

encoder – session serializer. A callable with the following signature: *def encode(param: Any) -> str: ...*. Default is `json.dumps()`.

decoder – session deserializer. A callable with the following signature: *def decode(param: str) -> Any: ...*. Default is `json.loads()`.

New in version 2.3: Added *encoder* and *decoder* parameters.

max_age
Maximum age for session data, `int` seconds or `None` for "session cookie" which last until you close your browser.

cookie_name
Name of cookie used for saving session data.

cookie_params
`dict` of cookie params: *domain*, *max_age*, *path*, *secure* and *httponly*.

encoder
The JSON serializer that will be used to dump session cookie data.

New in version 2.3.

decoder
The JSON deserializer that will be used to load session cookie data.

New in version 2.3.

new_session()
A coroutine for getting a new session regardless of whether a cookie exists.

Return *Session* instance.

load_session(*request*)
An *abstract* coroutine, called by internal machinery for retrieving *Session* object for given *request* (`aiohttp.web.Request` instance).

Return *Session* instance.

save_session(*request*, *response*, *session*)
An *abstract* coroutine, called by internal machinery for storing *session* (*Session*) instance for given *request* (`aiohttp.web.Request`) using *response* (`aiohttp.web.StreamResponse` or descendants).

load_cookie(*request*)
A helper for loading cookie (`http.cookies.SimpleCookie` instance) from *request* (`aiohttp.web.Request`).

save_cookie(*response*, *cookie_data*, *, *max_age=None*)
A helper for saving *cookie_data* (`str`) into *response* (`aiohttp.web.StreamResponse` or descendants).

max_age is cookie lifetime given from session. Storage defailt is used if the value is `None`.

Simple Storage

For testing purposes there is *SimpleCookieStorage*. It stores session data as unencrypted and unsigned JSON data in browser cookies, so it's totally insecure.

> **Warning:** Never use this storage on production!!! It's highly insecure!!!

To use the storage you should push it into *session_middleware()*:

```
aiohttp_session.setup(app, aiohttp_session.SimpleCookieStorage())
```

class aiohttp_session.**SimpleCookieStorage**(*, *cookie_name="AIOHTTP_SESSION"*, *domain=None*, *max_age=None*, *path='/'*, *secure=None*, *httponly=True*, *encoder=json.dumps*, *decoder=json.loads*)

Create unencrypted cookie storage.

The class is inherited from *AbstractStorage*.

Parameters are the same as for *AbstractStorage* constructor.

Cookie Storage

The storage that saves session data in HTTP cookies as `Fernet` encrypted data.

To use the storage you should push it into *session_middleware()*:

```
app = aiohttp.web.Application(middlewares=[
    aiohttp_session.cookie_storage.EncryptedCookieStorage(
        b'Thirty  two  length  bytes  key.'])
```

class aiohttp_session.cookie_storage.**EncryptedCookieStorage**(*secret_key*, *, *cookie_name="AIOHTTP_SESSION"*, *domain=None*, *max_age=None*, *path='/'*, *secure=None*, *httponly=True*, *encoder=json.dumps*, *decoder=json.loads*)

Create encryted cookies storage.

The class is inherited from *AbstractStorage*.

secret_key is `bytes` secret key with length of 32, used for encoding or base-64 encoded `str` one.

Other parameters are the same as for *AbstractStorage* constructor.

Note: For key generation use `cryptography.fernet.Fernet.generate_key()` method.

NaCl Storage

The storage that saves session data in HTTP cookies as `SecretBox` encrypted data.

To use the storage you should push it into *session_middleware()*:

```
app = aiohttp.web.Application(middlewares=[
    aiohttp_session.nacl_storage.NaClCookieStorage(
        b'Thirty  two  length  bytes  key.'])
```

class aiohttp_session.cookie_storage.**NaClCookieStorage**(*secret_key*, ***, *cookie_name="AIOHTTP_SESSION"*, *domain=None*, *max_age=None*, *path='/'*, *secure=None*, *httponly=True*, *encoder=json.dumps*, *decoder=json.loads*)

Create encryted cookies storage.

The class is inherited from *AbstractStorage*.

secret_key is `bytes` secret key with length of 32, used for encoding.

Other parameters are the same as for *AbstractStorage* constructor.

Redis Storage

The storage that stores session data in Redis database and keeps only Redis keys (UUIDs actually) in HTTP cookies.

It operates with Redis database via `aioredis.RedisPool`.

To use the storage you need setup it first:

```
redis = await aioredis.create_pool(('localhost', 6379))
storage = aiohttp_session.redis_storage.RedisStorage(redis)
aiohttp_session.setup(app, storage)
```

class aiohttp_session.redis_storage.**RedisStorage**(*redis_pool*, ***, *cookie_name="AIOHTTP_SESSION"*, *domain=None*, *max_age=None*, *path='/'*, *secure=None*, *httponly=True*, *key_factory=lambda: uuid.uuid4().hex*, *encoder=json.dumps*, *decoder=json.loads*)

Create Redis storage for user session data.

The class is inherited from *AbstractStorage*.

redis_pool is a `RedisPool` which should be created by `create_pool()` call, e.g.:

```
redis = await aioredis.create_pool(('localhost', 6379))
storage = aiohttp_session.redis_storage.RedisStorage(redis)
```

Other parameters are the same as for *AbstractStorage* constructor.

Memcahed Storage

The storage that stores session data in Memcached and keeps only keys (UUIDs actually) in HTTP cookies.

It operates with Memcahed database via `aiomecache.Client`.

To use the storage you need setup it first:

```
mc = aiomchache.Client('localhost', 11211)
storage = aiohttp_session.memcached_storage.Client(mc)
aiohttp_session.setup(app, storage)
```

New in version 1.2.

class aiohttp_session.redis_storage.**MemcachedStorage**(*memcached_conn*, ***, *cookie_name="AIOHTTP_SESSION"*, *domain=None*, *max_age=None*, *path='/'*, *secure=None*, *httponly=True*, *key_factory=lambda: uuid.uuid4().hex*, *encoder=json.dumps*, *decoder=json.loads*)

Create Memcached storage for user session data.

The class is inherited from *AbstractStorage*.

memcached_conn is a `Client` instance:

```
mc = await aiomcache.Client('localhost', 6379)
storage = aiohttp_session.memcached_storage.MemcachedStorage(mc
```

Other parameters are the same as for *AbstractStorage* constructor.

6.2 Glossary

aioredis *asyncio* compatible Redis client library

https://aioredis.readthedocs.io/

aiomcache *asyncio* compatible Memcached client library

https://github.com/aio-libs/aiomcache

asyncio The library for writing single-threaded concurrent code using coroutines, multiplexing I/O access over sockets and other resources, running network clients and servers, and other related primitives.

Reference implementation of **PEP 3156**

https://docs.python.org/3/library/asyncio.html

cryptography The libary used for encrypting secure cookied session

https://cryptography.io

pynacl Yet another libary used for encrypting secure cookied session

https://pynacl.readthedocs.io

session A namespace that is valid for some period of continual activity that can be used to represent a user's interaction with a web application.

sqlalchemy The Python SQL Toolkit and Object Relational Mapper.

http://www.sqlalchemy.org/

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index