Flask-Session

Welcome to Flask-Session's documentation. Flask-Session is an extension for Flask that adds support for Server-side `Session` to your application. Flask 0.8 or newer is required, if you are using an older version, check Support for Old and New Sessions out.

If you are not familiar with Flask, I highly recommend you to give it a try. Flask is a microframework for Python and it is really Fun to work with. If you want to dive into its documentation, check out the following links:

- Flask Documentation

Installation

Install the extension with the following command:

```
$ easy_install Flask-Session
```

or alternatively if you have pip installed:

```
$ pip install Flask-Session
```

Quickstart

Flask-Session is really easy to use.

Basically for the common use of having one Flask application all you have to do is to create your Flask application, load the configuration of choice and then create the **`Session`** object by passing it the application.

The `Session` instance is not used for direct access, you should always use **`flask.session`**:

```
from flask import Flask, session
from flask.ext.session import Session

app = Flask(__name__)
# Check Configuration section for more details
SESSION_TYPE = 'redis'
app.config.from_object(__name__)
Session(app)

@app.route('/set/')
def set():
    session['key'] = 'value'
    return 'ok'

@app.route('/get/')
def get():
    return session.get('key', 'not set')
```

You may also set up your application later using **`init_app()`** method:

```
sess = Session()
sess.init_app(app)
```

Configuration

The following configuration values exist for Flask-Session. Flask-Session loads these values from your Flask application config, so you should configure your app first before you pass it to Flask-Session. Note that these values cannot be modified after the `init_app` was applyed so make sure to not modify them at runtime.

We are not supplying something like `SESSION_REDIS_HOST` and `SESSION_REDIS_PORT`, if you want to use the `RedisSessionInterface`, you should configure `SESSION_REDIS` to your own `redis.Redis` instance. This gives you more flexibility, like maybe you want to use the same `redis.Redis` instance for cache purpose too, then you do not need to keep two `redis.Redis` instance in the same process.

The following configuration values are builtin configuration values within Flask itself that are related to session. **They are all understood by Flask-Session, for example, you should use PERMANENT_SESSION_LIFETIME to control your session lifetime.**

`SESSION_COOKIE_NAME`	the name of the session cookie
`SESSION_COOKIE_DOMAIN`	the domain for the session cookie. If this is not set, the cookie will be valid for all

	subdomains of `SERVER_NAME`.
`SESSION_COOKIE_PATH`	the path for the session cookie. If this is not set the cookie will be valid for all of `APPLICATION_ROOT` or if that is not set for `'/'`.
`SESSION_COOKIE_HTTPONLY`	controls if the cookie should be set with the httponly flag. Defaults to *True*.
`SESSION_COOKIE_SECURE`	controls if the cookie should be set with the secure flag. Defaults to *False*.
`PERMANENT_SESSION_LIFETIME`	the lifetime of a permanent session as **`datetime.timedelta`** object. Starting with Flask 0.8 this can also be an integer representing seconds.

A list of configuration keys also understood by the extension:

`SESSION_TYPE`	Specifies which type of session interface to use. Built-in session types: • **null**: NullSessionInterface (default) • **redis**: RedisSessionInterface • **memcached**: MemcachedSessionInterface • **filesystem**: FileSystemSessionInterface • **mongodb**: MongoDBSessionInterface • **sqlalchemy**: SqlAlchemySessionInterface
`SESSION_PERMANENT`	Whether use permanent session or not, default to be `True`
`SESSION_USE_SIGNER`	Whether sign the session cookie sid or not, if set to `True`, you have to set **`flask.Flask.secret_key`**, default to be `False`
`SESSION_KEY_PREFIX`	A prefix that is added before all session keys. This makes it possible to use the same backend storage server for different apps, default “session:”
`SESSION_REDIS`	A `redis.Redis` instance, default connect to `127.0.0.1:6379`
`SESSION_MEMCACHED`	A `memcache.Client` instance, default connect to `127.0.0.1:11211`
`SESSION_FILE_DIR`	The directory where session files are stored. Default to use *flask_session* directory under current working directory.
`SESSION_FILE_THRESHOLD`	The maximum number of items the session stores before it starts deleting some, default 500
`SESSION_FILE_MODE`	The file mode wanted for the session files, default 0600
`SESSION_MONGODB`	A `pymongo.MongoClient` instance, default connect to `127.0.0.1:27017`
`SESSION_MONGODB_DB`	The MongoDB database you want to use, default “flask_session”
`SESSION_MONGODB_COLLECT`	The MongoDB collection you want to use, default “sessions”
`SESSION_SQLALCHEMY`	A `flask.ext.sqlalchemy.SQLAlchemy` instance whose database connection URI is configured using the `SQLALCHEMY_DATABASE_URI` parameter
`SESSION_SQLALCHEMY_TABLE`	The name of the SQL table you want to use, default “sessions”

Basically you only need to configure `SESSION_TYPE`.

> Note:
>
> By default, all non-null sessions in Flask-Session are permanent.

New in version 0.2: `SESSION_TYPE`: **sqlalchemy**, `SESSION_USE_SIGNER`

Built-in Session Interfaces

`NullSessionInterface`

If you do not configure a different `SESSION_TYPE`, this will be used to generate nicer error messages. Will allow read-only access to the empty session but fail on setting.

`RedisSessionInterface`

Uses the Redis key-value store as a session backend. (redis-py required)

Relevant configuration values:

- SESSION_REDIS

`MemcachedSessionInterface`

Uses the Memcached as a session backend. (pylibmc or memcache required)

- SESSION_MEMCACHED

`FileSystemSessionInterface`

Uses the **`werkzeug.contrib.cache.FileSystemCache`** as a session backend.

- SESSION_FILE_DIR
- SESSION_FILE_THRESHOLD
- SESSION_FILE_MODE

`MongoDBSessionInterface`

Uses the MongoDB as a session backend. (pymongo required)

- SESSION_MONGODB
- SESSION_MONGODB_DB
- SESSION_MONGODB_COLLECT

`SqlAlchemySessionInterface`

New in version 0.2.

Uses SQLAlchemy as a session backend. (Flask-SQLAlchemy required)

- SESSION_SQLALCHEMY
- SESSION_SQLALCHEMY_TABLE

API

Changelog

Version 0.1

First public preview release.

Version 0.1.1

Fixed MongoDB backend InvalidDocument Error.

Version 0.2

- Added *SqlAlchemySessionInterface*.
- Added support for cookie session id signing.
- Various bugfixes.

Version 0.2.1

Fixed signing failure.

Version 0.2.2

Added support for non-permanent session.

Version 0.2.3

- Fixed signing failure in Python 3.x
- Fixed MongoDBSessionInterface failure in Python 3.x
- Fixed SqlAlchemySessionInterface failure in Python 3.x
- Fixed StrictRedis support

Version 0.3

- SqlAlchemySessionInterface is using LargeBinary type to store data now
- Fixed MongoDBSessionInterface delete method not found
- Fixed TypeError when getting store_id using a signer

Version 0.4

[Fork of orphaned repo at https://github.com/fengsp/flask-session to https://github.com/mcrowson/flask-session] - Add support for DynamoDB backend