gevent-socketio Documentation

Release 0.3.1

Jeffrey Gelens,Alex Bourget,John Anderson

March 19, 2015

CHAPTER 1

Introduction

Socket.IO is a WebSocket-like abstraction that enables real-time communication between a browser and a server. `gevent-socketio` is a Python implementation of the protocol.

The reference server implementation of Socket.IO runs on Node.js and was developed by LearnBoost. There are now server implementations in a variety of languages.

One aim of this project is to provide a single `gevent`-based API that works across the different WSGI-based web frameworks out there (Pyramid, Pylons, Flask, web2py, Django, etc...). Only ~3 lines of code are required to tie-in `gevent-socketio` in your framework. Note: you need to use the `gevent` python WSGI server to use `gevent-socketio`.

Namespaces: since you mostly have **one** websocket/socket.io endpoint per website, it is important to be able to namespace the different real-time activities of the different pages or parts of your site, just like you need routes to map URLs to different parts of your code. The Socket.IO 0.7+ namespaces are a welcome addition, and if you don't use Socket.IO, you'll probably end-up writing your own namespacing mechanism at some point.

Named events: To distinguish the messages that are coming and going, you most probably want to give them some name. Here again, not using Socket.IO, you will find yourself implementing a way to tag your packets with names representing different tasks or actions to perform. With Socket.IO 0.6 or with normal WebSockets, you would probably encode a JSON object with one of the keys that is reserved for that (I used `{"type": "submit_something"}`. Socket.IO 0.7+ implements named events, which put that information in a terse form on the wire. It also allows you to define callbacks, that can be acknowledged by the other endpoint, and then fire back your function with some return parameters. Something great for RPC, that you'd need to implement yourself the moment you need it.

Transports: One of the main feature of Socket.IO is the abstraction of the transport, that gives you real-time web support down to Internet Explorer 6.0, using long-polling methods. It will also use native WebSockets when available to the browser, for even lower latencies. Currently supported transports: `websocket`, `flashsocket`, `htmlfile`, `xhr-multipart`, `xhr-polling`, `jsonp-polling`.

This implementation covers nearly all the features of the Socket.IO 0.7+ (up to at least 0.9.1) protocol, with events, callbacks. It adds security in a pythonic way with granular ACLs (which don't exist in the Node.js version) at the method level. The project has several examples in the source code and in the documentation. Any addition and fixes to the docs are warmly welcomed.

CHAPTER 2

Concepts

In order to understand the following documentation articles, let's clarify some of the terms used:

A **Namespace** is like a controller in the MVC world. It encompasses a set of methods that are logically in it. For example, the `send_private_message` event would be in the `/chat` namespace, as well as the `kick_ban` event. Whereas the `scan_files` event would be in the `/filesystem` namespace. Each namespace is represented by a sub-class of `BaseNamespace`. A simple example would be, on the client side (the browser):

```
var socket = io.connect("/chat");
```

having loaded the `socket.io.js` library somewhere in your <head>. On the server (this is a Pyramid example, but its pretty much the same for other frameworks):

```
from socketio.namespace import BaseNamespace

class ChatNamespace(BaseNamespace):
    def on_chat(self, msg):
        self.emit('chat', msg)

def socketio_service(request):
    socketio_manage(request.environ, {'/chat': ChatNamespace},
                    request)
    return "out"
```

Here we use `socketio.socketio_manage()` to start the Socket.IO machine, and handle the real-time communication.

You will come across the notion of a `Socket`. This is a virtual socket, that abstracts the fact that some transports are long-polling and others are stateful (like a Websocket), and exposes the same functionality for all. You can have many namespaces inside a Socket, each delimited by their name like `/chat`, `/filesystem` or `/foobar`. Note also that there is a global namespace, identified by an empty string. Some times, the global namespace has special features, for backwards compatibilty reasons (we only have a global namespace in version 0.6 of the protocol). For example, disconnecting the global namespace means disconnect the full socket. Disconnecting a qualified namespace, on the other hand, only removes access to that namespace.

The `Socket` is responsible from taking the *packets*, which are, in the realm of a `Namespace` or a `Socket` object, a dictionary that looks like:

```
{"type": "event",
 "name": "launch_superhero",
 "args": ["Superman", 123, "km", {"hair_color": "brown"}]}
```

These packets are serialized in a compact form when its time to put them on the wire. Socket.IO also has some optimizations if we need to send many packets on some long-polling transports.

At this point, if you don't know `gevent`, you probably will want to learn a bit more about it, since it is the base you will be working on:

> http://www.gevent.org/

CHAPTER 3

Getting started

Until we have a fully-fledged tutorial, please check out our example applications and the API documentation.

You can see a video that shows `gevent-socketio` in a live coding presentation here:

> http://pyvideo.org/video/1573/gevent-socketio-cross-framework-real-time-web-li

To learn how to build your Namespace (the object dealing with requests and replies), see:

> *namespace_module*

See this doc for different servers integration:

> *server_integration*

CHAPTER 4

Examples

The `gevent-socketio` repository holds several examples:

> https://github.com/abourget/gevent-socketio/tree/master/examples
>
> - `simple_chat` is a bare-bone WSGI app with a minimal socketio integration
> - `simple_pyramid_chat` is a simple chat application built on Pyramid
> - `live_cpu_graph` is a simple realtime CPU graph (linux only)
> - `twitter_stream` is a streaming feed of twitter updates
> - `pyramid_backbone_redis_chat` is a Pyramid app using backbone.js and redis for pubsub
> - `pyramid_backbone_redis_chat_persistence` is a Pyramid app using backbone.js, redis for pubsub and features persistence
> - `testapp` is the app we use to test the different features, so there are a couple of more advanced use-cases demonstrated there

`pyvore` is an application that was developed to serve as real-time chat in conferences like the PyCon:

> https://github.com/sontek/pyvore

This app is a Django tic-tac-toe application that uses the latest `gevent-socketio`:

> https://github.com/sontek/django-tictactoe

CHAPTER 5

Security

gevent-socketio provides method-level security, using an ACL model. You can read more about it in the *namespace_module*, but a basic example to secure one namespace would look like:

```
class AdminInterface(BaseNamespace):
    def get_initial_acl(self):
        """Everything is locked at first"""
        return []

    def initialize(self):
        # This here assumes you have passed in a `request`
        # to your socketio_manage() call, it has that
        # `is_admin` attribute
        if not request.is_admin:
            return
        else:
            self.lift_acl_restrictions()

    def on_blahblahblah(self, data):
        """This can't be access until `lift_acl_restrictions()` has
        been called

        """
        pass
```

CHAPTER 6

API docs

API documentation is where most of the juice/meat is. Read through and you'll (hopefully) understand everything you need about `gevent-socketio`.

The manager is the function you call from your framework. It is in:

    socketio

Namespaces are the main interface the developer is going to use. You mostly define your own BaseNamespace derivatives, and gevent-socketio maps the incoming messages to your methods automatically:

    socketio.namespace

Mixins are components you can add to your namespaces, to provided added functionality.

    socketio.mixins

Sockets are the virtual tunnels that are established and abstracted by the different Transports. They basically expose socket-like send/receive functionality to the Namespace objects. Even when we use long-polling transports, only one Socket is created per browser.

    socketio.virtsocket

Packet is a library that handle the decoding of the messages encoded in the Socket.IO dialect. They take dictionaries for encoding, and return decoded dictionaries also.

    socketio.packet

Handler is a lower-level transports handler. It is responsible for calling your WSGI application

    socketio.handler

Transports are responsible for translating the different fallback mechanisms to one abstracted Socket, dealing with payload encoding, multi-message multiplexing and their reverse operation.

    socketio.transports

Server is the component used to hook Gevent and its WSGI server to the WSGI app to be served, while dispatching any Socket.IO related activities to the *handler* and the *transports*.

    socketio.server

Auto-generated indexes:

- *genindex*
- *modindex*

References

LearnBoost's node.js version is the reference implementation, you can find the server component at this address:

> https://github.com/learnboost/socket.io

The client JavaScript library's development branch is here:

> https://github.com/LearnBoost/socket.io-client

The specifications to the protocol are somehow in this repository:

> https://github.com/LearnBoost/socket.io-spec

This is the original wow-website:

> http://socket.io

Here is a list of the different frameworks integration to date, although not all have upgraded to the latest version of gevent-socketio:

- pyramid_socketio: https://github.com/abourget/pyramid_socketio
- django-socketio: https://github.com/stephenmcd/django-socketio

The Flask guys will be working on an integration layer soon.

CHAPTER 8

Contacts

For any questions, you can use the Issue tracking at Github:

> https://github.com/abourget/gevent-socketio https://github.com/abourget/gevent-socketio/issues

The mailing list:

> https://groups.google.com/forum/#!forum/gevent-socketio

The maintainers:

> https://twitter.com/bourgetalexndre https://twitter.com/sontek

Credits

Jeffrey Gellens for starting and polishing this project over the years.

PyCon 2012 and the Sprints, for bringing this project up to version 0.9 of the protocol.

Current maintainers:

- Alexandre Bourget
- John Anderson

Contributors:

- Denis Bilenko
- Bobby Powers
- Lon Ingram
- Eugene Baumstein
- Sébastien Béal
- jpellerin (JP)
- Philip Neustrom
- Jonas Obrist
- fabiodive
- Dan O'Neill
- Whit Morriss
- Chakib (spike) Benziane
- Vivek Venugopalan
- Vladimir Protasov
- Bruno Bigras
- Gabriel de Labacheliere
- Flavio Curella
- thapar
- Marconi Moreto
- sv1jsb

- Cliff Xuan
- Matt Billenstein
- Rolo
- Anthony Oliver
- Pierre Giraud
- m0sth8
- Daniel Swarbrick

CHAPTER 10

TODO

How to integrate your framework's "session" object (Beaker, memcached, or file-based). Beware: this can be tricky. You need to manage that yourself.