# sockjs-tornado documentation

***Release 0.2***

**Serge S. Koval**

April 02, 2015

Contents

This is implementation of the SockJS realtime transport library on top of the Tornado framework.

CHAPTER 1

# Topics

## 1.1 Statistics

sockjs-tornado captures some counters:

| Name | Description |
|---|---|
| **Sessions** | |
| sessions_active | Number of currently active sessions |
| **Transports** | |
| transp_xhr | # of sessions opened with xhr transport |
| transp_websocket | # of sessions opened with websocket transport |
| transp_xhr_streaming | # of sessions opened with xhr streaming transport |
| transp_jsonp | # of sessions opened with jsonp transport |
| transp_eventsource | # of sessions opened with eventsource transport |
| transp_htmlfile | # of sessions opened with htmlfile transport |
| transp_rawwebsocket | # of sessions opened with raw websocket transport |
| **Connections** | |
| connections_active | Number of currently active connections |
| connections_ps | Number of opened connections per second |
| **Packets** | |
| packets_sent_ps | Packets sent per second |
| packets_recv_ps | Packets received per second |

Stats are captured by the router object and can be accessed through the `stats` property:

```
MyRouter = SockJSRouter(MyConnection)

print MyRouter.stats.dump()
```

For more information, check stats module API or `stats` sample.

## 1.2 Frequently Asked Questions

### 1.2.1 How fast is sockjs-tornado?

Long story short, at least for websocket transport:

1. On CPython it is (or was) comparable to sockjs-node and socket.io (node.js server)
2. On PyPy 1.7 it was 3x faster than sockjs-node and socket.io

You can find more information here.

### 1.2.2 Can I use query string parameters or cookies to pass data to the server?

No, you can not. SockJS does not support application cookies and you can't pass query string parameters.

### 1.2.3 How do I implement authentication in my application?

Send packet with authentication token after connecting to the server. In pseudo code it can look like this:

Client side:

```
<script language="javascript">
    var sock = new SockJS('/echo');

    // Lets assume we invented simple protocol, where first word is a command name and second is a pa
    sock.onconnect = function() {
        sock.send('auth,xxx');
        sock.send('echo,bar');
    };
</script>
```

Server side:

```
class EchoConnection(SockJSConnection):
    def on_open(self, info):
        self.authenticated = False

    def on_message(self, msg):
        pack, data = msg.split(',', 1)

        # Ignore all packets except of 'auth' if user is not yet authenticated
        if not self.authenticated and pack != 'auth':
            return

        if pack == 'auth':
            # Decrypt user_id (or user name - you decide). You might want to add salt to the token as
            user_id = des_decrypt(data, secret_key)

            # Validate user_id here by making DB call, etc.
            user = get_user(user_id)

            if user is None and user.is_active:
                self.send('error,Invalid user!')
                return

            self.authenticated = True
        elif pack == 'echo':
            self.send(data)
```

### 1.2.4 Can I open more than one SockJS connection on same page?

No, you can not because of the AJAX restrictions. You have to use connection multiplexing. Check *multiplex* sample to find out how you can do it.

### 1.2.5 Can I send python objects through the SockJS?

As SockJS emulates websocket protocol and websocket protocol only supports strings, you can not send arbitrary objects (they won't be JSON serialized automatically). On a side node - SockJS client already includes *JSON2.js* library, so you can use it without any extra dependencies.

### 1.2.6 How do I scale sockjs-tornado?

Easiest way is to start multiple instances of the sockjs-tornado server (one per core) and load balance them using haproxy. You can find sample haproxy configuration here.

Alternatively, if you already use some kind of load balancer, make sure you enable sticky sessions. sockjs-tornado maintains state for each user and there's no way to synchronize this state between sockjs-tornado instances.

Also, it is up for you, as a developer, to decide how you're going to synchronize state of the servers in a cluster. You can use Redis as a meeting point or use ZeroMQ, etc.

## 1.3 API

### 1.3.1 `sockjs.tornado.conn`

#### sockjs.tornado.conn

SockJS connection interface

**class** `sockjs.tornado.conn.`**`SockJSConnection`**(*session*)

##### Callbacks

`SockJSConnection.`**`on_open`**(*request*)
Default on_open() handler.

Override when you need to do some initialization or request validation. If you return False, connection will be rejected.

You can also throw Tornado HTTPError to close connection.

***request*** `ConnectionInfo` object which contains caller IP address, query string parameters and cookies associated with this request (if any).

`SockJSConnection.`**`on_message`**(*message*)
Default on_message handler. Must be overridden in your application

`SockJSConnection.`**`on_close`**()
Default on_close handler.

##### Output

`SockJSConnection.`**`send`**(*message*, *binary=False*)
Send message to the client.

***message*** Message to send.

SockJSConnection.**broadcast**(*clients*, *message*)
Broadcast message to the one or more clients. Use this method if you want to send same message to lots of clients, as it contains several optimizations and will work fast than just having loop in your code.

***clients*** Clients iterable

***message*** Message to send.

#### Management

SockJSConnection.**close**()

SockJSConnection.**is_closed**
Check if connection was closed

### 1.3.2 sockjs.tornado.router

#### sockjs.tornado.router

SockJS protocol router implementation.

**class** sockjs.tornado.router.**SockJSRouter**(*connection*, *prefix=''*, *user_settings={}*, *io_loop=None*)
SockJS protocol router

SockJSRouter.**__init__**(*connection*, *prefix=''*, *user_settings={}*, *io_loop=None*)
Constructor.

***connection*** SockJSConnection class

***prefix*** Connection prefix

***user_settings*** Settings dictionary

***io_loop*** Optional IOLoop instance

#### URLs

SockJSRouter.**urls**
List of the URLs to be added to the Tornado application

SockJSRouter.**apply_routes**(*routes*)
Feed list of the URLs to the routes list. Returns list

#### Connection

SockJSRouter.**get_session**(*session_id*)
Get session by session id

***session_id*** Session id

SockJSRouter.**get_connection_class**()
Return associated connection class

### 1.3.3 sockjs.tornado.session

#### sockjs.tornado.session

SockJS session implementation.

##### Base Session

**class** sockjs.tornado.session.**BaseSession**(*conn*, *server*)
Base session implementation class

**Constructor**
BaseSession.**__init__**(*conn*, *server*)
Base constructor.

***conn*** Connection class

***server*** SockJSRouter instance

**Handlers**
BaseSession.**set_handler**(*handler*)
Set transport handler `handler`

Handler, should derive from the *sockjs.tornado.transports.base.BaseTransportMixin*.

BaseSession.**verify_state**()
Verify if session was not yet opened. If it is, open it and call connections *on_open*

BaseSession.**remove_handler**(*handler*)
Remove active handler from the session

***handler*** Handler to remove

**Messaging**
BaseSession.**send_message**(*msg*, *stats=True*, *binary=False*)
Send or queue outgoing message

***msg*** Message to send

***stats*** If set to True, will update statistics after operation completes

BaseSession.**send_jsonified**(*msg*, *stats=True*)
Send or queue outgoing message which was json-encoded before. Used by the *broadcast* method.

***msg*** JSON-encoded message to send

***stats*** If set to True, will update statistics after operation completes

BaseSession.**broadcast**(*clients*, *msg*)
Optimized *broadcast* implementation. Depending on type of the session, will json-encode message once and will call either *send_message* or *send_jsonifed*.

***clients*** Clients iterable

***msg*** Message to send

**State**

`BaseSession.close(code=3000, message='Go away!')`
Close session or endpoint connection.

***code*** Closing code

***message*** Close message

`BaseSession.delayed_close()`
Delayed close - won't close immediately, but on next ioloop tick.

`BaseSession.is_closed`
Check if session was closed.

`BaseSession.get_close_reason()`
Return last close reason tuple.

For example:

**if self.session.is_closed:** code, reason = self.session.get_close_reason()

### Connection Session

**class** `sockjs.tornado.session.Session(conn, server, session_id, expiry=None)`
SockJS session implementation.

**Constructor**

`Session.__init__(conn, server, session_id, expiry=None)`
Session constructor.

***conn*** Default connection class

***server*** *SockJSRouter* instance

***session_id*** Session id

***expiry*** Session expiry time

**Session**

`Session.on_delete(forced)`
Session expiration callback

***forced*** If session item explicitly deleted, forced will be set to True. If item expired, will be set to False.

**Handlers**

`Session.set_handler(handler, start_heartbeat=True)`
Set active handler for the session

***handler*** Associate active Tornado handler with the session

***start_heartbeat*** Should session start heartbeat immediately

`Session.verify_state()`
Verify if session was not yet opened. If it is, open it and call connections *on_open*

`Session.remove_handler(handler)`
Detach active handler from the session

***handler*** Handler to remove

**Messaging**

`Session.`**`send_message`**(*msg*, *stats=True*, *binary=False*)
Send or queue outgoing message

***msg*** Message to send

***stats*** If set to True, will update statistics after operation completes

`Session.`**`send_jsonified`**(*msg*, *stats=True*)
Send JSON-encoded message

***msg*** JSON encoded string to send

***stats*** If set to True, will update statistics after operation completes

`Session.`**`on_messages`**(*msg_list*)
Handle incoming messages

***msg_list*** Message list to process

**State**

`Session.`**`flush`**()
Flush message queue if there's an active connection running

`Session.`**`close`**(*code=3000*, *message='Go away!'*)
Close session.

***code*** Closing code

***message*** Closing message

**Heartbeats**

`Session.`**`start_heartbeat`**()
Reset hearbeat timer

`Session.`**`stop_heartbeat`**()
Stop active heartbeat

`Session.`**`delay_heartbeat`**()
Delay active heartbeat

`Session.`**`_heartbeat`**()
Heartbeat callback

### Utilities

**class** `sockjs.tornado.session.`**`ConnectionInfo`**(*ip*, *cookies*, *arguments*, *headers*, *path*)
Connection information object.

Will be passed to the `on_open` handler of your connection class.

Has few properties:

***ip*** Caller IP address

***cookies*** Collection of cookies

***arguments*** Collection of the query string arguments

***headers*** Collection of explicitly exposed headers from the request including: origin, referer, x-forward-for (and associated headers)

***path*** Request uri path

### 1.3.4 sockjs.tornado.migrate

#### sockjs.tornado.migrate

*tornado.websocket* to *sockjs.tornado* migration helper.

**class** sockjs.tornado.migrate.**WebsocketHandler**(*session*)

If you already use Tornado websockets for your application and want try sockjs-tornado, change your handlers to derive from this WebsocketHandler class. There are some limitations, for example only self.request only contains remote_ip, cookies and arguments collection

**open**()

open handler

**on_open**(*info*)

sockjs-tornado on_open handler

### 1.3.5 sockjs.tornado.proto

#### sockjs.tornado.proto

SockJS protocol related functions

#### JSON

Module contains two functions - *json_encode* and *json_decode* which you can use to encode/decode

sockjs.tornado.proto.**json_encode**(*data*)

sockjs.tornado.proto.**json_decode**(*data*)

#### SockJS protocol

SockJS protocol constants.

proto.**CONNECT = 'o'**

proto.**DISCONNECT = 'c'**

proto.**MESSAGE = 'm'**

proto.**HEARTBEAT = 'h'**

sockjs.tornado.proto.**disconnect**(*code*, *reason*)

Return SockJS packet with code and close reason

***code*** Closing code

***reason*** Closing reason

### 1.3.6 sockjs.tornado.basehandler

#### sockjs.tornado.basehandler

Various base http handlers

#### Base Request Handler

**class** sockjs.tornado.basehandler.**BaseHandler**(*application*, *request*, ***kwargs*)
Base request handler with set of helpers.

BaseHandler.**initialize**(*server*)
Initialize request

***server*** SockJSRouter instance.

**Stats**

BaseHandler.**prepare**()
Increment connection count

BaseHandler.**_log_disconnect**()
Decrement connection count

BaseHandler.**finish**(*chunk=None*)
Tornado *finish* handler

BaseHandler.**on_connection_close**()
Tornado *on_connection_close* handler

**Cache**

BaseHandler.**enable_cache**()
Enable client-side caching for the current request

BaseHandler.**disable_cache**()
Disable client-side cache for the current request

BaseHandler.**handle_session_cookie**()
Handle JSESSIONID cookie logic

**State**

BaseHandler.**safe_finish**()
Finish session. If it will blow up - connection was set to Keep-Alive and client dropped connection, ignore any IOError or socket error.

#### Preflight handler

**class** sockjs.tornado.basehandler.**PreflightHandler**(*application*, *request*, ***kwargs*)
CORS preflight handler

**Handler**

PreflightHandler.**options**(**args*, ***kwargs*)
XHR cross-domain OPTIONS handler

**Helpers**

PreflightHandler.**preflight**()
Handles request authentication

PreflightHandler.**verify_origin**()
Verify if request can be served

### 1.3.7 sockjs.tornado.periodic

#### sockjs.tornado.periodic

This module implements customized PeriodicCallback from tornado with support of the sliding window.

**class** sockjs.tornado.periodic.**Callback**(*callback*, *callback_time*, *io_loop*)
Custom implementation of the Tornado.Callback with support of callback timeout delays.

**__init__**(*callback*, *callback_time*, *io_loop*)
Constructor.

***callback*** Callback function

***callback_time*** Callback timeout value (in milliseconds)

***io_loop*** io_loop instance

**calculate_next_run**()
Caltulate next scheduled run

**start**(*timeout=None*)
Start callbacks

**stop**()
Stop callbacks

**delay**()
Delay callback

### 1.3.8 sockjs.tornado.sessioncontainer

#### sockjs.tornado.sessioncontainer

Simple heapq-based session implementation with sliding expiration window support.

**class** sockjs.tornado.sessioncontainer.**SessionMixin**(*session_id=None*, *expiry=None*)
Represents one session object stored in the session container. Derive from this object to store additional data.

**__init__**(*session_id=None*, *expiry=None*)
Constructor.

**session_id** Optional session id. If not provided, will generate new session id.

**expiry** Expiration time. If not provided, will never expire.

**is_alive**()
Check if session is still alive

**promote**()
Mark object as alive, so it won't be collected during next run of the garbage collector.

**on_delete**(*forced*)
Triggered when object was expired or deleted.

**class** sockjs.tornado.sessioncontainer.**SessionContainer**
Session container object.

If we will implement sessions with Tornado timeouts, for polling transports it will be nightmare - if load will be high, number of discarded timeouts will be huge and will be huge performance hit, as Tornado will have to clean them up all the time.

**add**(*session*)
Add session to the container.

***session*** Session object

**get**(*session_id*)
Return session object or None if it is not available

***session_id*** Session identifier

**remove**(*session_id*)
Remove session object from the container

***session_id*** Session identifier

**expire**(*current_time=None*)
Expire any old entries

***current_time*** Optional time to be used to clean up queue (can be used in unit tests)

### 1.3.9 sockjs.tornado.static

#### sockjs.tornado.static

Various static handlers required for SockJS to function properly.

**class** sockjs.tornado.static.**IFrameHandler**(*application*, *request*, ***kwargs*)
SockJS IFrame page handler

**class** sockjs.tornado.static.**GreetingsHandler**(*application*, *request*, ***kwargs*)
SockJS greetings page handler

**class** sockjs.tornado.static.**ChunkingTestHandler**(*application*, *request*, ***kwargs*)
SockJS chunking test handler

**class** sockjs.tornado.static.**InfoHandler**(*application*, *request*, ***kwargs*)
SockJS 0.2+ /info handler

### 1.3.10 sockjs.tornado.stats

**class** sockjs.tornado.stats.**StatsCollector**(*io_loop*)

**dump**()
Return dictionary with current statistical information

**class** sockjs.tornado.stats.**MovingAverage**(*period=10*)
Moving average class implementation

**__init__**(*period=10*)
Constructor.

***period*** Moving window size. Average will be calculated from the data in the window.

**add**(*n*)
Add value to the current accumulator

***n*** Value to add

**flush**()
Add accumulator to the moving average queue and reset it. For example, called by the StatsCollector once per second to calculate per-second average.

### 1.3.11 sockjs.tornado.transports.base

**class** sockjs.tornado.transports.base.**BaseTransportMixin**
Base transport.

Implements few methods that session expects to see in each transport.

**get_conn_info**()
Return *ConnectionInfo* object from current transport

**session_closed**()
Called by the session, when it gets closed

### 1.3.12 sockjs.tornado.transports.pollingbase

#### sockjs.tornado.transports.pollingbase

Polling transports base

**class** sockjs.tornado.transports.pollingbase.**PollingTransportBase**(*application*, *request*, ***kwargs*)
Polling transport handler base class

**send_message**(*message*, *binary=False*)
Called by the session when some data is available

**session_closed**()
Called by the session when it was closed

### 1.3.13 sockjs.tornado.transports.streamingbase

**class** sockjs.tornado.transports.streamingbase.**StreamingTransportBase**(*application*, *request*, ***kwargs*)

**should_finish**()
Check if transport should close long running connection after sending X bytes to the client.

***data_len*** Amount of data that was sent

**session_closed**()
Called by the session when it was closed

### 1.3.14 sockjs.tornado.transports.xhr

#### sockjs.tornado.transports.xhr

Xhr-Polling transport implementation

**class** sockjs.tornado.transports.xhr.**XhrPollingTransport**(*application*, *request*, ***kwargs*)

xhr-polling transport implementation

**post**(**args*, ***kwargs*)

**send_pack**(*message*, *binary=False*)

**class** sockjs.tornado.transports.xhr.**XhrSendHandler**(*application*, *request*, ***kwargs*)

**post**(*session_id*)

### 1.3.15 sockjs.tornado.transports.xhrstreaming

#### sockjs.tornado.transports.xhrstreaming

Xhr-Streaming transport implementation

**class** sockjs.tornado.transports.xhrstreaming.**XhrStreamingTransport**(*application*, *request*, ***kwargs*)

**post**(**args*, ***kwargs*)

**send_pack**(*message*, *binary=False*)

### 1.3.16 sockjs.tornado.transports.eventsource

#### sockjs.tornado.transports.eventsource

EventSource transport implementation.

**class** sockjs.tornado.transports.eventsource.**EventSourceTransport**(*application*, *request*, ***kwargs*)

**get**(**args*, ***kwargs*)

**send_pack**(*message*, *binary=False*)

### 1.3.17 sockjs.tornado.transports.jsonp

#### sockjs.tornado.transports.jsonp

JSONP transport implementation.

**class** sockjs.tornado.transports.jsonp.**JSONPTransport**(*application*, *request*, ***kwargs*)

**get**(**args*, ***kwargs*)

**send_pack**(*message*, *binary=False*)

**class** sockjs.tornado.transports.jsonp.**JSONPSendHandler**(*application*, *request*, ***kwargs*)

**post**(*session_id*)

### 1.3.18 sockjs.tornado.transports.htmlfile

#### sockjs.tornado.transports.htmlfile

HtmlFile transport implementation.

**class** sockjs.tornado.transports.htmlfile.**HtmlFileTransport**(*application*, *request*, ***kwargs*)

**get**(**args*, ***kwargs*)

**send_pack**(*message*, *binary=False*)

### 1.3.19 sockjs.tornado.transports.rawwebsocket

#### sockjs.tornado.transports.rawwebsocket

Raw websocket transport implementation

**class** sockjs.tornado.transports.rawwebsocket.**RawSession**(*conn*, *server*)

Raw session without any sockjs protocol encoding/decoding. Simply works as a proxy between *SockJSConnection* class and *RawWebSocketTransport*.

**send_message**(*msg*, *stats=True*, *binary=False*)

**on_message**(*msg*)

**class** sockjs.tornado.transports.rawwebsocket.**RawWebSocketTransport**(*application*, *request*, ***kwargs*)

Raw Websocket transport

**open**()

**on_message**(*message*)

**on_close**()

**send_pack**(*message*, *binary=False*)

**session_closed**()

**_detach**()

### 1.3.20 sockjs.tornado.transports.websocket

#### sockjs.tornado.transports.websocket

Websocket transport implementation

**class** sockjs.tornado.transports.websocket.**WebSocketTransport**(*application*, *request*, ***kwargs*)

Websocket transport

**open**(*session_id*)

**on_message**(*message*)

**on_close**()

**send_pack**(*message*, *binary=False*)

### Sessions

WebSocketTransport.**session_closed**()

WebSocketTransport.**_detach**()

CHAPTER 2

# Indices and tables

- *genindex*
- *modindex*
- *search*

## S

## Symbols



## A



## B



## C



## D



## E



## F



## G

## H



## I



## J



## M



## O



## P



## R

## S



## U



## V



## W



## X