jps Documentation

Release

Author

July 17, 2016

Contents

jps is simple pub/sub system for python. It is very easy to install and it is easy to understand if you have experience of ROS(Robot Operating System).

Contents:

CHAPTER 1

Introduction

jps(json pub/sub) is small wrapper of zeromq. It provides simple Pub/Sub system and command line tools, which is strongly inspired by ROS(Robot Operating System). jps is easier to install than ROS, and it does not have serialization. Let's use json format.

1.1 How to install

You can use pip to install jps

```
$ sudo pip install jps
```

It installs jps python module, `jps_master` and *jps_topic* command.

1.2 How to write pub/sub

publisher.py

```python
import jps
import time

pub = jps.Publisher('/hoge1')
i = 0
while True:
  pub.publish('hello! jps{0}'.format(i))
  i += 1
  time.sleep(0.5)
```

subscriber.py

```python
import jps

def callback(msg):
  print msg

sub = jps.Subscriber('/hoge1', callback)
sub.spin()
```

1.3 How to run and use tools

You need three consoles to test the program.

```
$ jps_master
$ python publisher.py
$ python subscriber.py
```

To get the list of the topics, you can use jps_topic list

```
$ jps_topic list
```

If you want to see the data in /hoge1 topic,

```
$ jps_topic echo /hoge1
```

CHAPTER 2

ActionServer and ActionClient

Do you know about actionlib of ROS? jps provide simple ActionServer and ActionClient. If you want to provide some action which takes long time, how about using these classes. It is made by pub/sub only, but it is possible to handle the response correctly, because it manage what is the response of the request.

Below is a sample of ActionServer

```
import jps
import time

def callback(req):
    time.sleep(1)
    print req + ' received'
    return True

s = jps.ActionServer('move_to', callback)
s.spin()
```

Below is a sample of ActionClient

```
import jps
import json
import time

c = jps.ActionClient('move_to')
time.sleep(0.1) # need this sleep
future = c(json.dumps({'x': 10.0, 'y': 0.1}))
print 'do something during waiting response'
time.sleep(1)
result = future.wait()
print result
```

It does not contain feedback topic, it is the difference between ROS.

Simple Subscriber

If your subscriber is simple you can write it easier than ROS style. Simple means below here.

- subscribe only one topic
- just while loop main function

simple_subscriber

```
import jps


for msg in jps.Subscriber('/hoge1'):
    print msg
```

This prints /hoge1 messages.

You can mix the two styles if you want.

```
import jps

def callback(msg):
    print 'hoge2 = {}'.format(msg)

sub2 = jps.Subscriber('/hoge2', callback)
for msg in jps.Subscriber('/hoge1'):
    print 'hoge1 is here!{}'.format(msg)
    sub2.spin_once()
```

CHAPTER 4

jps_topic

jsp_topic is similar tool to rostopic. It is automatically installed by `pip install jps`.

If you want to know how to use, try `-h` option.

```
$ jps_topic -h
usage: jps_topic [-h] {pub,echo,list,record,play} ...

json pub/sub tool

positional arguments:
  {pub,echo,list,record,play}
                        command
    pub                 publish topic from command line
    echo                show topic data
    list                show topic list
    record              record topic data
    play                play recorded topic data

optional arguments:
  -h, --help            show this help message and exit
```

For all commands, you can set the host of *jps_master* by *–host HOST*, if you want to use jps from remote computer.

4.1 jps_topic pub

pub command publishes json text data.

```
$ jps_topic pub topic_name "{\"data\": 1.0}"
```

It publishes only once if the `--repeat` option is not specified. see `-h` option for more detail.

```
$ jps_topic pub -h
usage: jps_topic pub [-h] [--host HOST] [--publisher_port PUBLISHER_PORT] [--repeat REPEAT] topic_nar

positional arguments:
  topic_name            name of topic
  data                  json string data to be published

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           master host
  --publisher_port PUBLISHER_PORT, -p PUBLISHER_PORT
```

```
                        publisher port
  --repeat REPEAT, -r REPEAT
                        repeat in hz
```

4.2 jps_topic echo

echo command prints json text data.

```
$ jps_topic echo -h
usage: jps_topic echo [-h] [--host HOST] [--subscriber_port SUBSCRIBER_PORT] [--num NUM] topic_name

positional arguments:
  topic_name         name of topic

optional arguments:
  -h, --help         show this help message and exit
  --host HOST           master host
  --subscriber_port SUBSCRIBER_PORT, -s SUBSCRIBER_PORT
                        subscriber port
  --num NUM, -n NUM  print N times and exit
```

4.3 jps_topic list

list command collects all topics and create list of topic names, which is published now. It subscribes 1[sec] to create the list. If the topic is published less than 1[Hz], the list comman may not catch the name. You can use `--timeout` option to catch the slow topics.

```
$ jps_topic list --help
usage: jps_topic list [-h] [--host HOST] [--subscriber_port SUBSCRIBER_PORT] [--timeout TIMEOUT]

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           master host
  --subscriber_port SUBSCRIBER_PORT, -s SUBSCRIBER_PORT
                        subscriber port
  --timeout TIMEOUT, -t TIMEOUT
                        timeout in sec
```

4.4 jps_topic record

record command is like rosbag command. It records the topic data to the file. You can replay the data by `play` command. You can use `--file` option to specify the output file name. Default is `record.json`. You can set the topic name to be recorded. If the topic_names is empty, all topics will be recorded. Because the file format is normal json, you can read/parse it by any json reader if you want.

```
$ jps_topic record -h
usage: jps_topic record [-h] [--host HOST] [--subscriber_port SUBSCRIBER_PORT] [--file FILE] [topic_

positional arguments:
  topic_names           topic names to be recorded
```

```
optional arguments:
  -h, --help            show this help message and exit
  --host HOST           master host
  --subscriber_port SUBSCRIBER_PORT, -s SUBSCRIBER_PORT
                        subscriber port
  --file FILE, -f FILE  output file name (default: record.json)
```

4.5 jps_topic play

play command replays the saved data by `jps_topic record`.

```
$ jps_topic play -h
usage: jps_topic play [-h] [--host HOST] [--publisher_port PUBLISHER_PORT] file

positional arguments:
  file                  input file name

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           master host
  --publisher_port PUBLISHER_PORT, -p PUBLISHER_PORT
                        publisher port
```

CHAPTER 5

Environmental variables

jps uses below environmental variables. These are optional. You don't need to set these variables (default will be used.)

- *JPS_MASTER_HOST*: set default master host. (default: "localhost")
- *JPS_SUFFIX*: Add this to all topic names. This is for multi robot system. (default: "")
- *JPS_MASTER_PUB_PORT*: port number for publishers (default: 54320)
- *JPS_MASTER_SUB_PORT*: port number for subscribers (default: 54321)
- *JPS_SERIALIZE*: default serialzier. Only 'json' is supported. (default: None)
- *JPS_REMAP*: remap topic names. If you set 'export JPS_REMAP="hoge=foo"', topic 'hoge' will be changed to 'foo'

Serializer

jps itsself does not have serializer. If you want to serialize your payload, You have to handle yourself.

Only serialization by json is supported. If you set environ like below (for bash),

```
export JPS_SERIALIZE=json
```

payload will be automatically serialized as json. Actually, it does just `json.dumps(payload)` before published, `json.loads(payload)` after subscription.

You can use any serializer if you pass it to publisher/subscriber.

```
def my_serialize(payload):
  return payload + 'hoge'

def my_deserialize(payload):
  return payload + 'hoge'

pub = jps.Publisher('topic1', serializer=my_serialize)
sub = jps.Subscriber('topic1', serializer=my_deserialize)
```

CHAPTER 7

jps package

7.1 Module contents

class jps.**Publisher**(*topic_name*, *host=None*, *pub_port=None*, *serializer='DEFAULT'*)

Bases: `object`

Publishes data for a topic.

Example:

```
>>> pub = jps.Publisher('special_topic')
>>> pub.publish('{"name": "hoge"}')
```

Parameters

- **topic_name** – Topic name
- **host** – host of subscriber/forwarder
- **pub_port** – port of subscriber/forwarder
- **serializer** – this function is applied before publish (default: None)

publish(*payload*)

Publish payload to the topic

Note: If you publishes just after creating Publisher instance, it will causes lost of message. You have to add sleep if you just want to publish once.

```
>>> pub = jps.Publisher('topic')
>>> time.sleep(0.1)
>>> pub.publish('{data}')
```

Parameters **payload** – data to be published. This is ok if the data is not json.

class jps.**Subscriber**(*topic_name*, *callback=None*, *host=None*, *sub_port=None*, *deserializer='DEFAULT'*)

Bases: `object`

Subscribe the topic and call the callback function

Example:

```
>>> def callback(msg):
...   print msg
...
>>> sub = jps.Subscriber('topic_name', callback)
>>> sub.spin()
```

or you can use python generator style

```
>>> import jps
>>> for msg in jps.Subscriber('/hoge1'):
...   print msg
```

Parameters

- **`topic_name`** – topic name
- **`host`** – host name of publisher/forwarder
- **`sub_port`** – port of publisher/forwarder
- **`deserializer`** – this function is applied after received (default: None)

`deserialize`(*msg*)

`next`()
receive next data (block until next data)

`spin`(*use_thread=False*)
call callback for all data forever (until C-c)

Parameters **`use_thread`** – use thread for spin (do not block)

`spin_once`(*polling_sec=0.01*)
Read the queued data and call the callback for them. You have to handle KeyboardInterrupt (C-c) manually.

Example:

```
>>> def callback(msg):
...   print msg
>>> sub = jps.Subscriber('topic_name', callback)
>>> try:
...   while True:
...     sub.spin_once():
...     time.sleep(0.1)
... except KeyboardInterrupt:
...   pass
```

class `jps.`**`ArgumentParser`**(*subscriber=True*, *publisher=True*, *service=False*, **args*, ***kwargs*)
Bases: `argparse.ArgumentParser`

Create ArgumentParser with args (host/subscriber_port/publisher_port)

Example:

```
>>> parser = jps.ArgumentParser(description='my program')
>>> args = parser.parse_args()
>>> args.host
'localhost'
>>> args.subscriber_port
54321
>>> args.publisher_port
54320
```

Parameters

- **add subscriber_port (default** (*subscriber*) – True)
- **add publisher_port (default** (*publisher*) – True)

class jps.**Authenticator**(*public_keys_dir*)
Bases: object

classmethod instance(*public_keys_dir*)
Please avoid create multi instance

set_client_key(*zmq_socket*, *client_secret_key_path*, *server_public_key_path*)
must call before bind

set_server_key(*zmq_socket*, *server_secret_key_path*)
must call before bind

stop()

jps.**create_certificates**(*keys_dir='certificates'*)

class jps.**ServiceServer**(*callback*, *host=None*, *res_port=None*, *use_security=False*)
Bases: object

Example:

```
>>> def callback(req):
...   return 'req = {req}'.format(req=req)
...
>>> service = jps.ServiceServer(callback)
>>> service.spin()
```

close()

spin(*use_thread=False*)
call callback for all data forever (until C-c)

Parameters use_thread – use thread for spin (do not block)

spin_once()

class jps.**ServiceClient**(*host=None*, *req_port=None*, *use_security=False*)
Bases: object

call(*request*)

class jps.**ActionServer**(*base_topic_name*, *callback*, *host=None*, *pub_port=None*, *sub_port=None*, *serializer='DEFAULT'*, *deserializer='DEFAULT'*)
Bases: object

serve the service which takes some long time

Example:

```
>>> import jps
>>> import time
>>> def callback(req):
...   time.sleep(1)
...   return req + ' received'
>>> s = jps.ActionServer('move_to', callback)
# subscribe 'move_to/request', publish 'move_to/response'
>>> s.spin()
```

spin(*use_thread=False*)

spin_once()

class jps.**ActionClient**(*base_topic_name*, *host=None*, *pub_port=None*, *sub_port=None*, *serializer='DEFAULT'*, *deserializer='DEFAULT'*)

Bases: `object`

Call an action

Example:

```
>>> import jps
>>> import json
>>> c = jps.ActionClient('move_to')
>>> future = c(json.dumps({'x': 10.0, 'y': 0.1}))
# do something if you are busy to do something during waiting.
>>> result = future.wait()
```

class jps.**Bridge**(*upload_topic_names*, *download_topic_names*, *remote_host=None*, *remote_pub_port=None*, *remote_sub_port=None*)

Bases: `object`

spin()

class jps.**BridgeServiceServer**(*download_topics*, *sub_port=None*, *pub_port=None*, *res_port=None*, *use_security=False*)

Bases: `jps.bridge.BridgeServiceBase`

callback(*request*)

close()

spin(*use_thread=False*)

class jps.**BridgeServiceClient**(*upload_topics*, *frequency=10.0*, *sub_port=None*, *pub_port=None*, *host=None*, *req_port=None*, *use_security=False*)

Bases: `jps.bridge.BridgeServiceBase`

spin(*use_thread=False*)

spin_once()

exception jps.**Error**

Bases: `exceptions.Exception`

7.2 Submodules

7.3 jps.utils module

class jps.utils.**JsonMultiplePublisher**

Bases: `object`

publish multiple topics by one json message

Example:

```
>>> p = JsonMultiplePublisher()
>>> p.publish('{"topic1": 1.0, "topic2": {"x": 0.1}}')
```

publish(*json_msg*)

json_msg = '{"topic1": 1.0, "topic2": {"x": 0.1}}'

class `jps.utils.`**`MultiplePublisher`**(*base_topic_name*)
Bases: `object`

`publish`(*msg*, *topic_suffix=’‘*)

7.4 jps.launcher module

`jps.launcher.`**`get_launched_module_pid_file`**(*module_name*)

`jps.launcher.`**`kill_module`**(*module_name*)

`jps.launcher.`**`launch_modules`**(*module_names*, *module_args={}*, *kill_before_launch=True*)

`jps.launcher.`**`launch_modules_with_names`**(*modules_with_names*, *module_args={}*, *kill_before_launch=True*)
launch module.main functions in another process

7.5 jps.args module

class `jps.args.`**`ArgumentParser`**(*subscriber=True*, *publisher=True*, *service=False*, **args*, ***kwargs*)
Bases: `argparse.ArgumentParser`

Create ArgumentParser with args (host/subscriber_port/publisher_port)

Example:

```
>>> parser = jps.ArgumentParser(description='my program')
>>> args = parser.parse_args()
>>> args.host
'localhost'
>>> args.subscriber_port
54321
>>> args.publisher_port
54320
```

Parameters

- **`add subscriber_port (default`** (*`subscriber`*) – True)
- **`add publisher_port (default`** (*`publisher`*) – True)

7.6 jps.tools module

`jps.tools.`**`echo`**(*topic_name*, *num_print=None*, *out=<open file ‘<stdout>’, mode ‘w’>*, *host=’localhost’*, *sub_port=54321*)
print the data for the given topic forever

`jps.tools.`**`play`**(*file_path*, *host=’localhost’*, *pub_port=54320*)
replay the recorded data by record()

`jps.tools.`**`pub`**(*topic_name*, *json_msg*, *repeat_rate=None*, *host=’localhost’*, *pub_port=54320*)
publishes the data to the topic

Parameters

- **`topic_name`** – name of the topic

- **`json_msg`** – data to be published
- **`repeat_rate`** – if None, publishes once. if not None, it is used as [Hz].

`jps.tools.`**`record`**(*file_path*, *topic_names=[]*, *host='localhost'*, *sub_port=54321*)
record the topic data to the file

`jps.tools.`**`show_list`**(*timeout_in_sec*, *out=<open file '<stdout>', mode 'w'>*, *host='localhost'*, *sub_port=54321*)
get the name list of the topics, and print it

`jps.tools.`**`topic_command`**()
command line tool for jps

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index