PySyncObj Documentation

Release 0.3.4

Filipp Ozinov

Sep 09, 2018

Contents

- The code is available on GitHub at bakwc/PySyncObj

Contents:

CHAPTER 1

pysyncobj package

1.1 SyncObj

class pysyncobj.**SyncObj**(*selfNodeAddr*, *otherNodesAddrs*, *conf=None*, *consumers=None*)

Main SyncObj class, you should inherit your own class from it.

Parameters

- **selfNodeAddr** (*str*) – address of the current node server, 'host:port'
- **otherNodesAddrs** (*list of str*) – addresses of partner nodes, ['host1:port1', 'host2:port2', ...]
- **conf** (SyncObjConf) – configuration object
- **consumers** (*list of SyncObjConsumer inherited objects*) – objects to be replicated

addNodeToCluster(*nodeName*, *callback=None*)

Add single node to cluster (dynamic membership changes). Async. You should wait until node successfully added before adding next node.

Parameters

- **nodeName** (*str*) – nodeHost:nodePort
- **callback** (function(*FAIL_REASON*, None)) – will be called on success or fail

destroy()

Correctly destroy SyncObj. Stop autoTickThread, close connections, etc.

doTick(*timeToWait=0.0*)

Performs single tick. Should be called manually if *autoTick* disabled

Parameters **timeToWait** (*float*) – max time to wait for next tick. If zero - perform single tick without waiting for new events. Otherwise - wait for new socket event and return.

forceLogCompaction()

Force to start log compaction (without waiting required time or required number of entries)

getStatus()
Dumps different debug info about cluster to dict and return it

isReady()
Check if current node is initially synced with others and has an actual data.

Returns True if ready, False otherwise

Return type bool

printStatus()
Dumps different debug info about cluster to default logger

removeNodeFromCluster(*nodeName*, *callback=None*)
Remove single node from cluster (dynamic membership changes). Async. You should wait until node successfully added before adding next node.

Parameters

- **nodeName** (*str*) – nodeHost:nodePort
- **callback** – will be called on success or fail

setCodeVersion(*newVersion*, *callback=None*)
Switch to a new code version on all cluster nodes. You should ensure that cluster nodes are updated, otherwise they won't be able to apply commands.

Parameters **newVersion** – new code version

:type int :param callback: will be called on cussess or fail :type callback: function(*FAIL_REASON*, None)

waitBinded()
Waits until initialized (binded port). If success - just returns. If failed to initialized after conf.maxBindRetries - raise SyncObjException.

1.2 replicated

pysyncobj.**replicated**(**decArgs*, ***decKwargs*)
Replicated decorator. Use it to mark your class members that modifies a class state. Function will be called asynchronously. Function accepts flowing additional parameters (optional):

'callback': callback(result, failReason), failReason - *FAIL_REASON*. 'sync': True - to block execution and wait for result, False - async call. If callback is passed,

'sync' option is ignored.

'timeout': if 'sync' is enabled, and no result is available for 'timeout' seconds -
SyncObjException will be raised.

These parameters are reserved and should not be used in kwargs of your replicated method.

Parameters

- **func** (*function*) – arbitrary class member
- **ver** (*int*) – (optional) - code version (for zero deployment)

1.3 replicated_sync

pysyncobj.**replicated_sync**(**decArgs*, ***decKwargs*)

1.4 SyncObjConf

class pysyncobj.**SyncObjConf**(***kwargs*)

PySyncObj configuration object

appendEntriesBatchSizeBytes = None
Max number of bytes per single append_entries command.

appendEntriesPeriod = None
Interval of sending append_entries (ping) command. Should be less than raftMinTimeout.

appendEntriesUseBatch = None
Send multiple entries in a single command. Enabled (default) - improve overall performance (requests per second) Disabled - improve single request speed (don't wait till batch ready)

autoTick = None
Disable autoTick if you want to call onTick manually. Otherwise it will be called automatically from separate thread.

bindAddress = None
Bind address (address:port). Default - None. If None - selfAddress is used as bindAddress. Could be useful if selfAddress is not equal to bindAddress. Eg. with routers, nat, port forwarding, etc.

bindRetryTime = None
Will try to bind port every bindRetryTime seconds until success.

commandsQueueSize = None
Commands queue is used to store commands before real processing.

commandsWaitLeader = None
If true - commands will be enqueued and executed after leader detected. Otherwise - *FAIL_REASON.MISSING_LEADER* error will be emitted. Leader is missing when esteblishing connection or when election in progress.

connectionRetryTime = None
Interval between connection attempts. Will try to connect to offline nodes each connectionRetryTime.

connectionTimeout = None
When no data received for connectionTimeout - connection considered dead. Should be more than raftMaxTimeout.

deserializer = None
Custom deserialize function, it will be called when restore from fullDump. If specified - there should be a custom serializer too. Should return data - internal stuff that was passed to serialize.

dnsCacheTime = None
Time to cache dns requests (improves performance, no need to resolve address for each connection attempt).

dnsFailCacheTime = None
Time to cache failed dns request.

dynamicMembershipChange = None
If enabled - cluster configuration could be changed dynamically.

fullDumpFile = None
File to store full serialized object. Save full dump on disc when doing log compaction. None - to disable store.

journalFile = None
File to store operations journal. Save each record as soon as received.

leaderFallbackTimeout = None
When leader has no response from the majority of the cluster for leaderFallbackTimeout - it will fallback to follower state. Should be more than appendEntriesPeriod.

logCompactionBatchSize = None
Max number of bytes per single append_entries command while sending serialized object.

logCompactionMinEntries = None
Log will be compacted after it reach minEntries size or minTime after previous compaction.

logCompactionMinTime = None
Log will be compacted after it reach minEntries size or minTime after previous compaction.

logCompactionSplit = None
If true - each node will start log compaction in separate time window. eg. node1 in 12.00-12.10, node2 in 12.10-12.20, node3 12.20 - 12.30, then again node1 12.30-12.40, node2 12.40-12.50, etc.

maxBindRetries = None
Max number of attempts to bind port (default 0, unlimited).

onCodeVersionChanged = None
This callback will be called when cluster is switched to new version. onCodeVersionChanged(oldVer, newVer)

onReady = None
This callback will be called as soon as SyncObj sync all data from leader.

onStateChanged = None
This callback will be called for every change of SyncObj state. Arguments: onStateChanged(oldState, newState). WARNING: there could be multiple leaders at the same time!

pollerType = None
Sockets poller: * *auto* - auto select best available on current platform * *select* - use select poller * *poll* - use poll poller

preferredAddrType = None
Preferred address type. Default - ipv4. None - no preferences, select random available. ipv4 - prefer ipv4 address type, if not available us ipv6. ipv6 - prefer ipv6 address type, if not available us ipv4.

raftMaxTimeout = None
Same as raftMinTimeout

raftMinTimeout = None
After randomly selected timeout (in range from minTimeout to maxTimeout) leader considered dead, and leader election starts.

recvBufferSize = None
Size of receive for sockets.

sendBufferSize = None
Size of send buffer for sockets.

serializeChecker = None
Check custom serialization state, for async serializer. Should return one of *SERIALIZER_STATE*.

serializer = None
Custom serialize function, it will be called when logCompaction (fullDump) happens. If specified - there should be a custom deserializer too. Arguments: serializer(fileName, data) data - some internal stuff that is *required* to be serialized with your object data.

useFork = None
Use fork if available when serializing on disk.

1.5 FAIL_REASON

class pysyncobj.**FAIL_REASON**

DISCARDED = 3
Command discarded (cause of new leader elected and another command was applied instead)

LEADER_CHANGED = 5
Simmilar to NOT_LEADER - leader has changed without command commit.

MISSING_LEADER = 2
Leader is currently missing (leader election in progress, or no connection)

NOT_LEADER = 4
Leader has changed, old leader did not have time to commit command.

QUEUE_FULL = 1
Commands queue full

REQUEST_DENIED = 6
Command denied

SUCCESS = 0
Command successfully applied.

1.6 SERIALIZER_STATE

class pysyncobj.**SERIALIZER_STATE**

FAILED = 3
Serialization failed (should be returned only one time after finished).

NOT_SERIALIZING = 0
Serialization not started or already finished.

SERIALIZING = 1
Serialization in progress.

SUCCESS = 2
Serialization successfully finished (should be returned only one time after finished).

CHAPTER 2

pysyncobj.batteries package

2.1 ReplCounter

class pysyncobj.batteries.**ReplCounter**

Simple distributed counter. You can set, add, sub and inc counter value.

add(**args*, ***kwargs*)

Adds value to a counter.

Parameters **value** – value to add

Returns new counter value

get()

Returns current counter value

inc(**args*, ***kwargs*)

Increments counter value by one.

Returns new counter value

set(**args*, ***kwargs*)

Set new value to a counter.

Parameters **newValue** – new value

Returns new counter value

sub(**args*, ***kwargs*)

Subtracts a value from counter.

Parameters **value** – value to subtract

Returns new counter value

2.2 ReplList

class pysyncobj.batteries.**ReplList**
Distributed list - it has an interface similar to a regular list.

append(**args*, ***kwargs*)
Append item to end

count(*element*)
Return number of occurrences of element

extend(**args*, ***kwargs*)
Extend list by appending elements from the iterable

get(*position*)
Return value at given position

index(*element*)
Return first position of element. Raises ValueError if the value is not present.

insert(**args*, ***kwargs*)
Insert object before position

pop(**args*, ***kwargs*)
Remove and return item at position (default last). Raises IndexError if list is empty or index is out of range.

rawData()
Return internal list - use it carefully

remove(**args*, ***kwargs*)
Remove first occurrence of element. Raises ValueError if the value is not present.

reset(**args*, ***kwargs*)
Replace list with a new one

set(**args*, ***kwargs*)
Update value at given position.

sort(**args*, ***kwargs*)
Stable sort *IN PLACE*

2.3 ReplDict

class pysyncobj.batteries.**ReplDict**
Distributed dict - it has an interface similar to a regular dict.

clear(**args*, ***kwargs*)
Remove all items from dict

get(*key*, *default=None*)
Return value for given key, return default if key not exist

items()
Return all items

keys()
Return all keys

pop(**args*, ***kwargs*)
Remove and return value for given key, return default if key not exist

rawData()
Return internal dict - use it carefully

reset(**args*, ***kwargs*)
Replace dict with a new one

set(**args*, ***kwargs*)
Set value for specified key

setdefault(**args*, ***kwargs*)
Return value for specified key, set default value if key not exist

update(**args*, ***kwargs*)
Adds all values from the other dict

values()
Return all values

2.4 ReplSet

class pysyncobj.batteries.**ReplSet**
Distributed set - it has an interface similar to a regular set.

add(**args*, ***kwargs*)
Add an element to a set

clear(**args*, ***kwargs*)
Remove all elements from this set.

discard(**args*, ***kwargs*)
Remove an element from a set if it is a member. If the element is not a member, do nothing.

pop(**args*, ***kwargs*)
Remove and return an arbitrary set element. Raises KeyError if the set is empty.

rawData()
Return internal dict - use it carefully

remove(**args*, ***kwargs*)
Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.

reset(**args*, ***kwargs*)
Replace set with a new one

update(**args*, ***kwargs*)
Update a set with the union of itself and others.

2.5 ReplQueue

class pysyncobj.batteries.**ReplQueue**(*maxsize=0*)
Replicated FIFO queue. Based on collections.deque. Has an interface similar to Queue.

Parameters **maxsize** (*int*) – Max queue size.

empty()
True if queue is empty

full()
True if queue is full

get(**args*, ***kwargs*)
Extract item from queue. Return default if queue is empty.

put(**args*, ***kwargs*)
Put an item into the queue. True - if item placed in queue. False - if queue is full and item can not be placed.

qsize()
Return size of queue

2.6 ReplPriorityQueue

class pysyncobj.batteries.**ReplPriorityQueue**(*maxsize=0*)
Replicated priority queue. Based on heapq. Has an interface similar to Queue.

Parameters **maxsize** (*int*) – Max queue size.

empty()
True if queue is empty

full()
True if queue is full

get(**args*, ***kwargs*)
Extract the smallest item from queue. Return default if queue is empty.

put(**args*, ***kwargs*)
Put an item into the queue. Items should be comparable, eg. tuples. True - if item placed in queue. False - if queue is full and item can not be placed.

qsize()
Return size of queue

2.7 ReplLockManager

class pysyncobj.batteries.**ReplLockManager**(*autoUnlockTime*, *selfID=None*)
Replicated Lock Manager. Allow to acquire / release distributed locks.

Parameters

- **autoUnlockTime** (*float*) – lock will be released automatically if no response from holder for more than autoUnlockTime seconds
- **selfID** (*str*) – (optional) - unique id of current lock holder.

destroy()
Destroy should be called before destroying ReplLockManager

isAcquired(*lockID*)
Check if lock is acquired by ourselves.

Parameters **lockID** (*str*) – unique lock identifier.

:return True if lock is acquired by ourselves.

release(*lockID*, *callback=None*, *sync=False*, *timeout=None*)
Release previously-acquired lock.

Parameters

- **lockID** (*str*) – unique lock identifier.
- **sync** (*bool*) – True - to wait until lock is released or failed to release.
- **callback** (*func(opResult, error)*) – if sync is False - callback will be called with operation result.
- **timeout** (*float*) – max operation time (default - unlimited)

tryAcquire(*lockID*, *callback=None*, *sync=False*, *timeout=None*)

Attempt to acquire lock.

Parameters

- **lockID** (*str*) – unique lock identifier.
- **sync** (*bool*) – True - to wait until lock is acquired or failed to acquire.
- **callback** (*func(opResult, error)*) – if sync is False - callback will be called with operation result.
- **timeout** (*float*) – max operation time (default - unlimited)

:return True if acquired, False - somebody else already acquired lock

CHAPTER 3

Indices and tables

- genindex
- search

Index

W