tf.train.LooperThread

Class **`LooperThread`**

Defined in `tensorflow/python/training/coordinator.py`.

See the guide: Training > Coordinator and QueueRunner

A thread that runs code repeatedly, optionally on a timer.

This thread class is intended to be used with a `Coordinator`. It repeatedly runs code specified either as `target` and `args` or by the `run_loop()` method.

Before each run the thread checks if the coordinator has requested stop. In that case the looper thread terminates immediately.

If the code being run raises an exception, that exception is reported to the coordinator and the thread terminates. The coordinator will then request all the other threads it coordinates to stop.

You typically pass looper threads to the supervisor `Join()` method.

Properties

`daemon`

A boolean value indicating whether this thread is a daemon thread (True) or not (False).

This must be set before start() is called, otherwise RuntimeError is raised. Its initial value is inherited from the creating thread; the main thread is not a daemon thread and therefore all threads created in the main thread default to daemon = False.

The entire Python program exits when no alive non-daemon threads are left.

`ident`

Thread identifier of this thread or None if it has not been started.

This is a nonzero integer. See the thread.get_ident() function. Thread identifiers may be recycled when a thread exits and another thread is created. The identifier is available even after the thread has exited.

`name`

A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

Methods

`__init__`

```
__init__(
    coord,
    timer_interval_secs,
    target=None,
    args=None,
    kwargs=None
)
```

Create a LooperThread.

Args:

- `coord`: A Coordinator.
- `timer_interval_secs`: Time boundaries at which to call Run(), or None if it should be called back to back.
- `target`: Optional callable object that will be executed in the thread.
- `args`: Optional arguments to pass to `target` when calling it.
- `kwargs`: Optional keyword arguments to pass to `target` when calling it.

Raises:

- `ValueError`: If one of the arguments is invalid.

`getName`

```
getName()
```

`isAlive`

```
isAlive()
```

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

`isDaemon`

```
isDaemon()
```

`is_alive`

```
is_alive()
```

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

`join`

```
join(timeout=None)
```

Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call isAlive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

`loop`

```
@staticmethod
loop(
    coord,
    timer_interval_secs,
    target,
    args=None,
    kwargs=None
)
```

Start a LooperThread that calls a function periodically.

If **`timer_interval_secs`** is None the thread calls **`target(args)`** repeatedly. Otherwise **`target(args)`** is called every **`timer_interval_secs`** seconds. The thread terminates when a stop of the coordinator is requested.

Args:

- `coord`: A Coordinator.
- `timer_interval_secs`: Number. Time boundaries at which to call **`target`**.
- `target`: A callable object.
- `args`: Optional arguments to pass to **`target`** when calling it.
- `kwargs`: Optional keyword arguments to pass to **`target`** when calling it.

Returns:

The started thread.

`run`

```
run()
```

run_loop

```
run_loop()
```

Called at 'timer_interval_secs' boundaries.

setDaemon

```
setDaemon(daemonic)
```

setName

```
setName(name)
```

start

```
start()
```

Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

start_loop

```
start_loop()
```

Called when the thread starts.

stop_loop

```
stop_loop()
```

Called when the thread stops.