Overview

Thespian
Python Actor System

By: Kevin Quick <kquick@godaddy.com>

2015 Sep 05 (#1.2)

Thespian Project

TheDoc-05

PUBLIC DOCUMENT

Contents

1 Home

Introducing Thespian

Thespian is a Python library providing a framework for developing concurrent, distributed, fault tolerant applications.

Thespian is built on the Actor Model which allows applications to be written as a group of indpendently executing but cooperating "Actors" which communicate via messages. These Actors run within the Actor System provided by the Thespian library.

Concurrent All Actors run independently within the Actor System. The Actor System may run the Actors as threads, processes, or even sequential operations within the current process—all with no change to the Actors themselves.

Distributed Actors run independently... anywhere. Multiple servers can each be running Thespian and an Actor can be run on any of these systems—all with no change to the Actors themselves. Thespian handles the communication between the Actors and the management process of distributing the Actors across the systems.

Location Independent Because Actors run independently anywhere, they run independently of their actual location. A distributed Actor application may have part of it running on a local server, part running on a server in Amsterdam, and part running on a server in Singapore... or not, with no change or awareness of this by the Actors themselves.

Fault Tolerant Individual Actors can fail and be restarted—automatically—without impact to the rest of the system.

Scalable The number of Actors in the system can be dynamically extended based on factors such as work volume, and systems added to the Distributed Actor System environment are automatically utilized.

One of the key aspects of the Actor Model is that it represents a higher level of abstraction than is provided by most frameworks. When writing an Actor-based application, the concurrency and transport layers are completely abstracted, which both simplifies the design and allows the concurrency or transport to be changed in the future *without* requiring changes in the Actor-based application.

The above qualities of Actor programming make it ideally suited for Cloud-based applications as well, where compute nodes are added and removed from the environment dynamically.

2 Quick Start

2.1 Simple Installation

Install the Thespian library on your local host system using pip:

```
$ pip install thespian
```

2.2 Starting a Thespian Instance

Start a default Thespian Actor System on the current host system:

```
$ python
>>> from thespian import *
>>> ActorSystem("multiprocTCPBase")
```

2.3 Hello World

A Hello World example:

```
 1  from thespian.actors import *
 2
 3  class Hello(Actor):
 4      def receiveMessage(self, message, sender):
 5          self.send(sender, 'Hello, World!')
 6
 7  if __name__ == "__main__":
 8      hello = ActorSystem().createActor(Hello)
 9      print(ActorSystem().ask(hello, 'hi', 1))
10      ActorSystem().tell(hello, ActorExitRequest())
```

The above will create an Actor, send it a message and get a "Hello World" response, and then tell that Actor to exit because it is no longer needed.

```
$ python helloActor.py
Hello, World!
$
```

2.4 Hello World Redux

It's easy to extend the Hello World example to include multiple actors which communicate with each other. This example shows a number of additional details:

- the messages exchanged between the Actors can be anything that can be pickled.
- Actors can create other Actors dynamically
- Actor Addresses can be passed around
- Actors remain until they are removed by sending them an `ActorExitRequest()`

```
 1  from thespian.actors import *
 2
 3  import sys
 4
 5  class Greeting(object):
 6      def __init__(self, msg):
 7          self.message = msg
 8      def __str__(self): return self.message
 9
10  class Hello(Actor):
11      def receiveMessage(self, message, sender):
12          if message == 'hi':
13              greeting = Greeting('Hello')
14              world = self.createActor(World)
15              punct = self.createActor(Punctuate)
```

```
16              greeting.sendTo = [punct, sender]
17              self.send(world, greeting)
18
19  class World(Actor):
20      def receiveMessage(self, message, sender):
21          if isinstance(message, Greeting):
22              message.message = message.message + ", World"
23              nextTo = message.sendTo.pop(0)
24              self.send(nextTo, message)
25
26  class Punctuate(Actor):
27      def receiveMessage(self, message, sender):
28          if isinstance(message, Greeting):
29              message.message = message.message + "!"
30              nextTo = message.sendTo.pop(0)
31              self.send(nextTo, message)
32
33  if __name__ == "__main__":
34      hello = ActorSystem().createActor(Hello)
35      print(ActorSystem().ask(hello, 'hi', 0.2))
36      print(ActorSystem().ask(hello, 'hi', 0.2))
37      ActorSystem().tell(hello, ActorExitRequest())
38      print(ActorSystem().ask(hello, 'hi', 0.2))
```

Running above will create an Actor and send it a message. That Actor will create two other Actors, passing the message along to the first which then passes it to the second before finally sending the message back to the original requestor.

The original requestor is code outside of the Actor environment. This external requestor uses the `ask()` API call which assigns it an Actor Address just like any other Actor.

```
$ python helloActor.py
Hello, World!
$
```

This is just a quick introduction to getting started with Thespian. Please see the examples section for more information.

2.5 What Next?

This has just been a very simple introduction to using Thespian. A more detailed introduction can be found in the Thespian In-Depth Introduction document, and Using Thespian is the primary API and utilization document.

3 Documentation

3.1 Thespian Documentation

TXT	PDF	HTML
TXT	PDF	Using Thespian is the main documentation reference for developers writing and implementing Actor-based applicati
TXT	PDF	Thespian Developer's Notes provides documentation for developers working on Thespian itself.
TXT	PDF	Release History is maintained in the Thespian Release Notes.
TXT	PDF	An In Depth Introduction to Thespian.

3.2 Background and Related Efforts

- The Actor Model at Wikipedia
- `http://c2.com/cgi/wiki?ActorsModel`
- Akka is a popular Actor Library for Java and Scala; it is roughly the equivalent to Thespian for those languages.
- Akka.NET is an implementation of Akka for the .NET and Mono environments making it useable for C# and F# applications.
- The Erlang language has Actor support built-in (see `http://www.erlang.org/doc/getting_started/conc_prog.html`).

4 Download

The recommended way to obtain Thespian is by using `pip` to download the latest copy from PyPi:

```
$ pip install thespian
```

The PyPi page can be consulted directly at `https://pypi.python.org/pypi/thespian`

Thespian source and releases are also available from the `https://github.com/godaddy/Thespian` github source maintenance location.

Thespian may also be provided by your system's packaging tools (e.g. yum, rpm, apt-get, etc.).

5 Contribute

Contributions to and involvement in Thespian development itself is welcomed. The Thespian Developer's Notes document provides more details on how to get involved.

- Source Code: `https://github.com/godaddy/Thespian`
- Mailing List: thespianpy@googlegroups.com
- Report a Bug: `https://github.com/godaddy/Thespian/issues`

6 News

2015-08-31 Thespian Publicly Released

Current and previous Thespian release information is available in the Thespian Release Notes.

6.1 Blogs and Success Stories

- GoDaddy blog article

If you have a blog, article, or success story you would like published here, please contact one of the project administrators.