ToiletPaper #143

jambit
WHERE INNOVATION WORKS

Aktor-System für Python

Autor: Hannes Lerchl / Senior Software Architect / New Business

✗ Problem

Aktor-Systeme sind schon eine ganze Weile bekannt (1970er, witzigerweise im Rahmen der AI-Forschung) und bieten eine zuverlässige Methode für hochgradig parallele Programmausführung. Die bekanntesten Vertreter sind heute wahrscheinlich Erlang, Elixir und Akka. Ach, gäbe es doch so etwas auch, um die häufiger werdenden Python-Backend-Anfragen zu bearbeiten.

✓ Lösung

Look over there! In der Tat gibt es auch für Python ein Aktor-System: Thespian. Eine ausgereifte und gut dokumentierte Bibliothek, die einen vernünftigen Grundstock vorgibt:

- „Actor Spawning“ und „Message Passing“ wird übernommen.
- Über Delayed Messages kann man Timeouts realisieren.
- Im Hintergrund wird für jeden Aktor ein eigener OS-Prozess erstellt, was die Sache dann zwar etwas schwergewichtiger macht, als Nebeneffekt aber auch den GIL umschifft.
- Aktoren können per Message beendet werden (damit wird dann auch der Prozess beendet).
- Log-Messages werden eingesammelt und zusammengeführt.
- Mit dem `ActorTypeDispatcher` gibt es eine Basisklasse, die einem schon mal die empfangenen Nachrichten nach Typ sortiert und spezialisierte (zu schreibende) Methoden aufruft.
- Das System kann auch Datei-Deskriptoren und Sockets „überwachen“ (per `select`), ohne, dass der ganze Aktor blockiert.
- Es gibt ein „Dead Letter Handling“ (für Messages, deren Empfänger bereits verschwunden ist).
- Es gibt eine Rückmeldung (als Message), wenn die eigene Message einen anderen Aktor zum Stolpern (sprich: exception) gebracht hat.
- Man kann Aktoren unter einem Namen global registrieren (mit den üblichen Nebenwirkungen).
- Über das Feature „Actor Troupes“ ist eine horizontale Skalierung von Aktoren möglich.
- Shell-Kommandos können über einen Aktor ausgeführt und der Output rückgemeldet werden (quasi ein Wrapper um `subprocess.Popen`).

➔ Beispiel

```python
from thespian.actors import *

class Hello(Actor):
  def receiveMessage(self, message, sender):
    # a message could be anything that python can pickle/unpickle
    self.send(sender, 'Hello, World!')

    # the actor keeps running and listening for messages

if __name__ == "__main__":
  # the actor system is created as well as a new actor.
  # this actor will run as its own OS-process on the same machine
  hello = ActorSystem().createActor(Hello)

  # send a message to the actor and wait (1 second) for a response
  print(ActorSystem().ask(hello, 'hi', 1))

  # command the actor to exit
  ActorSystem().tell(hello, ActorExitRequest())
```

+ Weiterführende Aspekte

- Wovon redet der gute Mann? Vom actor model.
- Doku findet sich im Thespian user's guide.
- Als Carl Hewitt mit seiner Schreibmaschine das actor model erfand: „Viewing control structures as patterns of passing messages“.
- Was leider fehlt, ist ein Ökosystem für „generische pattern“, wie es etwa Erlang/OTP bietet.