```
jobim
```

An actors library for Clojure modelled after Erlang OTP plaform.

External dependencies

Jobim has two dependencies:

- Directory service: ZooKeeper
- Messaging system: RabbitMQ/ZeroMQ

RabbitMQ or ZeroMQ both can be used as the underlying messaging solution. In order to use Jobim, ZeroMQ java bindings must be installed, including the native jni library.

Basic usage

starting a node : jobim/bootstrap-node

A path to a configuration file must be provided. The path must include the name of the node, the selected messaging mechanism and configuration options for RabbitMQ, ZeroMQ and ZooKeeper.

This is a sample configuration file:

```
;; RabbitMQ used
{:node-name "remote-test"
 :messaging-type :rabbitmq
 :messaging-options {:host "192.168.1.35"}
 :zookeeper-options ["192.168.1.35:2181" {:timeout 3000}]}

 This is a configuration file for ZeroMQ:

{:node-name "remote-test"
 :messaging-type :zeromq
 :messaging-options {:protocol-and-port "tcp://192.168.1.35:5555"}
 :zookeeper-options ["localhost:2181" {:timeout 3000}]}
```

Checking available nodes : jobim/nodes jobim/resolve-node-name

The nodes function returns a map with all the node names and their identifiers. The function resolve-node-name can be used to retrieve the indetifier of a node provided its name.

> => (use 'jobim)

> nil

> => (bootstrap-node "node-config.clj")

> "6811651bd83e4d428359b419e7f76a75"

> => (nodes)

> {"osx" "5299491ea4184c02ad8c0fbc100c49f9", "linux" "6811651bd83e4d428359b419e7f76a75"}

Agent creation : jobim/spawn

We can define an actor using any Clojure function and the spawn function.

```
; A Sample echo actor
(defn ping
  ([]
     (loop [continue true
            msg (receive)]
       (cond-match
        [#"exit" msg]       (recur false msg)
        [#"exception" msg]  (throw (Exception.
                                     (str "Ping actor with PID:"
                                          (self)
                                          "received exception")))
        [[?from ?data] msg] (do (send! from data)
                            (recur true (receive))))))
```

Creation of the actor:

> => (def *pid* (spawn examples/ping))

> #'clojure.core/*pid*

> => *pid*

> "5299491ea4184c02ad8c0fbc100c49f9.1"

Transforming a thread into an actor : jobim/spawn-in-repl

The REPL or any other thread can become an actor calling spawn-in-repl from the code being executed in the thread:

```
> => (spawn-in-repl)
> "5299491ea4184c02ad8c0fbc100c49f9.2"
> => (send! *pid* [(self) 13123])
> :ok
> => (receive)
> 13123
```

Sending and receiving : jobim/send! jobim/receive

This couple of function can be used to send and receive messages to and from actors. The PID of the actor must be provided. By default, any serializable java object can be send and retrieved.

```
> => (send! pid [(self) (java.util.Date.)]) > :ok
> => (receive)
> #<Date Fri Sep 03 13:20:26 CEST 2010>
```

RPC calls : jobim/rpc-call jobim/rpc-blocking-call

Used to execute a function call in a remote node. They can be used to spawn remote actors and retrieve the PID (using the blocking variant).

```
> => (nodes)
> {"osx" "5299491ea4184c02ad8c0fbc100c49f9", "linux" "6811651bd83e4d428359b419e7f76a75"}
> => (resolve-node-name "linux")
> "6811651bd83e4d428359b419e7f76a75"
> => (rpc-blocking-call (resolve-node-name "linux") "clojure.core/+" [1 2 3 4 5])
> 15
```

Spawning a remote actor:

```
> => (def *pid* (rpc-blocking-call (resolve-node-name "linux") "jobim/spawn" ["jobim.examples.actors/ping"]))
> #'clojure.core/*pid*
> => *pid*
> "6811651bd83e4d428359b419e7f76a75.1"
> => (send! *pid* [(self) 345])
> nil
> => (receive)
> 345
```

Registering PIDs : jobim/register-name jobim/registered-names jobim/resolve-name

An actor can be registered with a globally available name using the register-name function. All registered names can be retrieved with the registered-names function.

To transform an actor name into a PID the resolve-name function can be used.

```
> => (def *ping* (spawn examples/ping))
> #'clojure.core/*ping*
> => *ping*
> "5299491ea4184c02ad8c0fbc100c49f9.8"
> => (register-name "ping" *ping*)
> :ok
> => (registered-names)
> {"ping" "5299491ea4184c02ad8c0fbc100c49f9.8"}
> => (resolve-name "ping")
> "5299491ea4184c02ad8c0fbc100c49f9.8"
> => (send! (resolve-name "ping") [(self) 1234])
> :ok
```

> => (receive)

> 1234

Error notifications : jobim/link

Two actors can be linked using the link function. From that moment onwards if one of the actors dies because of an exception or the node where it is being executed becomes unavailable due to a network partition, the other actor will receive a special message containing a signal.

> => (self)

> "5299491ea4184c02ad8c0fbc100c49f9.1"

> => (def *pid* (spawn examples/ping))

> #'clojure.core/*pid*

> => (link *pid*)

> {"5299491ea4184c02ad8c0fbc100c49f9.1" ["5299491ea4184c02ad8c0fbc100c49f9.9"], > "5299491ea4184c02ad8c0fbc100c49f9.9" ["5299491ea4184c02ad8c0fbc100c49f9.1"]}

> => ; the ping actor will throw an exception if receives a message containing the

> "exception" string

> => (send! *pid* "exception")

> :ok

> => (receive)

> {:signal :link-broken, :from "5299491ea4184c02ad8c0fbc100c49f9.9", > :cause "class java.lang.Exception:Ping actor received exception"}

Threadless actors : jobim/spawn-evented jobim/react-loop jobim/react

To avoid the limitation on the number of java threads that can be created in a single java VM, an evented actor can be created not attached to a single thread.

Functions react-loop and react can be used as subtitues of loop and receive in the code of the evented actor.

The evented actor must be started with the spawn-evented function.

This is an example of the evented actor equivalent to the previously shown echo actor.

```
(defn ping-evented
  ([]
     (let [name "test evented"]
       (react-loop [c 0]
         (react [msg]
           (cond-match
             [#"exit" msg]       false
             [#"exception" msg]  (throw (Exception. "Ping actor received exception"))
             [[?from ?data] msg] (do
                                   (send! from (str "actor " name " says "data " " c))
                                   (recur (inc c)))))))))
```

Tests

To run the tests follow the next instructions:

- edit the test-node.clj file inserting the right configuration.
- edit test/jobim/test/jobim.clj and select the right communication mechanism:

  (defonce messaging-to-test :rabbitmq)
- Start the relevant services: ZooKeeper and RabbitMQ/ZeroMQ
- Start the test node that will execute the test-node.clj configuration:

> $ java -cp jobim-0.0.4-SNAPSHOT-standalone.jar -Djava.library.path=/usr/local/lib jobim.main test-node.clj

> ** Jobim node started **

> - node-name: remote-test

> - messaging-type :rabbitmq

> - messaging-args {:host "192.168.1.35"}

> - zookeeper-args ["192.168.1.35:2181" {:timeout 3000}

> clojure.core=>

- Run the tests using

> $ lein test

License