

Agent

The Agent is a special-purpose thread-safe non-blocking reference implementation inspired by Agents in Clojure.

Introduction

Agents safe-guard mutable values by allowing only a single **agent-managed thread** to make modifications to them. The mutable values are **not directly accessible** from outside, but instead **requests have to be sent to the agent** and the agent guarantees to process the requests sequentially on behalf of the callers. Agents guarantee sequential execution of all requests and so consistency of the values.

Agents are asynchronous active objects that accept code (functions) as messages. After reception the function is run against the internal state of the Agent and the return value of the function is considered to be the new internal state of the Agent.

Schematically:

```
agent = new Agent(0) //created a new Agent
wrapping an integer with initial value 0
agent.send {increment()} //asynchronous
send operation, sending the increment()
function
...
//after some delay to process the message
the internal Agent's state has been updated
...
assert agent.val== 1
```

To wrap integers, we can certainly use AtomicXXX types of the Java platform. When the state or the update algorithms become more complex we need more support.

Examples

```
import groovyx.gpars.agent.Agent

/**
 * Conference stores number of registrations
 and allows parties to register and
```

```
unregister.
```

```
 * It inherits from the Agent class and adds  
the register() and unregister() private  
methods,
```

```
 * which callers may use it the commands  
they submit to the Conference.
```

```
 */
```

```
class Conference extends Agent<Long> {  
    def Conference() { super(0) }  
    private def register(long num) { data +=  
num }  
    private def unregister(long num) { data  
-= num }  
}
```

```
final Agent conference = new Conference()  
//new Conference created
```

```
/**
```

```
 * Three external parties will try to  
register/unregister concurrently
```

```
 */
```

```
final Thread t1 = Thread.start {  
    conference << {register(10L)}  
//send a command to register 10 attendees  
}
```

```
final Thread t2 = Thread.start {  
    conference << {register(5L)}  
//send a command to register 5 attendees
```

```
}
```

```
final Thread t3 = Thread.start {  
    conference << {unregister(3L)}  
    //send a command to unregister 3 attendees  
}
```

```
[t1, t2, t3]*.join()  
assert 12L == conference.val
```

For latest update, see [the Agent section of the User Guide](#) and the respective [Demos](#).