luerl_sandbox(3)

Robert Virding

2023

Name

luerl_sandbox - Fuctions for sandboxing Luerl evaluation

Interface Functions

The **Lua State** parameter is the state of a Lua VM instance. It must be created with the **luerl:init()** call and be carried from one call to the next.

As it is possible in Lua to create self-referencing data structures, indeed the standard libraries have many instances of this, then using the functions which decode their return values will generate an error when they would cause an infinite loop during the decoding. An simple example is the top level table which contains a key `_G` which references the top-level table.

Note that Lua **Chunks** (see definition below) can travel between different States. They are precompiled bits of code, independent of State. That you can 'carry around' this is no unique to Luerl but a low-level implementation detail of the standard Lua language, for more on chunks read the official Lua 5.3 reference manual.

Spec Definitions

Binary means an Erlang binary string.
Chunks means a portion of precompiled bytecode.
State means a Lua State, this *is* a Lua VM instance.
Path means a file system path and file name.
KeyPath means an Erlang list of **atoms** representing nested names, e.g. [table,pack] for table.pack.
Keys means Lua table keys, the keys of a key-value structure.

Functions

init() -> State.

init([State | TablePaths]) -> State.

init(State, TablePaths) -> State. Create a new sandboxed state. If a state is given as an argument then that state will be used otherwise a new default be be generated. `TablePaths` is a list of paths to functions which will be blocked. If none is given then the default list will be used.

run(String | Binary) -> {Result, State} | {error, Reason}.

run(String | Binary, State) -> {Result, State} | {error, Reason}.

run(String | Binary, Flags, State) -> {Result, State} | {error, Reason}. Spawn a new process which runs the string `String` in `State` where the default sandbox state will be used if none is given. `Flags` is a map or keyword list which can contain the following fields

```
#{max_time => MaxTime,
  max_reductions => MaxReds,
  spawn_opts => SpawnOpts}
```

`MaxReds` limits the number of reductions and `MaxTime` (default 100 msecs) the time to run the string, `SpawnOpts` are spawn options to the process running the evaluation.

run(String | Binary) -> {Result, State} | {error, Reason}.

run(String | Binary, State) -> {Result, State} | {error, Reason}.

run(String | Binary, State, [MaxReds | Flags]) -> {Result, State} | {error, Reason}.

run(String | Binary, State, MaxReds, Flags) -> {Result, State} | {error, Reason}.

run(String | Binary, State, MaxReds, Flags, Timeout) -> {Result, State} | {error, Reason}. This is the old interface to run. It still works but the new interface is recommended. Spawn a new process which runs the string `String` in `State` where the default sandbox state will be used if none is given. `MaxReds` limits the number of reductions and `TimeOut` (default 100 msecs) the time to run the string, `Flags` are spawn options to the process running the evaluation.