v8-cffi Documentation

Release 0.1

Esteban Castro Borsani

August 07, 2016

Contents

CHAPTER 1

User's Guide

1.1 Installation

1.1.1 Compatibility

- Python 2.7, 3.4, 3.5 and PyPy 5.3

Note: **Linux-x64** is the only (officially) supported platform. To build the binaries for other platforms, the `./dev` steps must be adapted (probably to vagrant instead of docker) accordingly. PRs are welcome.

1.1.2 Pip

Latest version can be installed through pip:

```
$ pip install v8-cffi
```

1.2 Usage

1.2.1 Quick-start

```
from v8cffi import shortcuts
shortcuts.set_up()

ctx = shortcuts.get_context()
ctx.load_libs(['./foo_bundled.js'])
ctx.run_script('foo.render("hola mundo");')
# "hola mundo"
```

This is the most simple and limited form of usage. A platform, VM and global context are created when calling `set_up`, then the global context can be retrieve by calling `get_context` from anywhere within the application, even from other threads.

The normal usage is to `set_up` and `load_libs` at the start of the application, then call `run_script` in many other places.

One thing to have in mind is the context is *global*, this means any modification to the global scope will persist.

Tip: The API section contains useful information about: *platform*, *vm* and *context*.

1.2.2 Multiple VMs & contexts

```
from v8cffi.platform import platform

with platform as p:
    with p.create_vm() as vm:
        with vm.create_context() as ctx:
            ctx.load_libs(['./foo_bundled.js'])
            res = ctx.run_script('foo.render("hola mundo");')

        with vm.create_context() as ctx_b:
            ctx_b.load_libs(['./bar.js', './baz.js'])
            res_b = ctx_b.run_script('baz.render("hello world");')

    with p.create_vm() as vm_b:
        # ...
```

Or alternatively, without `with` statement:

```
from v8cffi.platform import platform

p = platform
p.set_up()
vm = p.create_vm()
vm.set_up()
ctx = vm.create_context()

try:
    # ...
finally:
    ctx.tear_down()
    vm.tear_down()
    p.tear_down()
```

This allows much greater flexibility than the previous `shortcuts` example.

As you may have read in the API section, the first thing is to create a V8 platform, there must be one and only one per process (python instance).

Creating many VMs allows to run JS code in parallel. Every VM takes about ~1MB of RAM. It should be possible to spawn a fixed pool of them with loaded libs to reuse and obtain N times the number of opts/s (where N is the number of VMs).

Every VM should run in a different python thread, so when calling `run_script`, the thread blocks (the GIL is released) and another python thread can run code in a different VM.

In case two python threads are using the *same* VM, the VM will prevent them from running at the same time. But both will block and release the GIL, allowing other python threads to run.

At last, creating many Context allows JS code run without sharing the same global scope.

CHAPTER 2

API Reference

Information on a specific function, class or method.

2.1 API

2.1.1 Platform Object

class v8cffi.platform.**_Platform**

V8 platform environment. The underlying platform is a singleton that must only be initialized once per process.

Should be used through *platform*

Variables

- **natives_path** (*str*) – Path to natives_blob.bin
- **snapshot_path** (*str*) – Path to snapshot_blob.bin

create_vm()

Create a *VM* for running JS scripts within an isolated environment

Returns Instance of *VM*

Return type *VM*

is_alive()

Check is initialized and was not exited

Returns Whether the platform is alive or not

Return type bool

set_up()

Initialize the V8 platform. Remember to call *tear_down()* before exiting the application. It's recommended to use a `with` statement instead of this method to ensure clean up.

This must only be called once in an application lifetime

Raises **V8MemoryError** – if there is no memory for allocating it, the process should die afterwards anyway, there is little point in catching this

tear_down()

Destructs the V8 platform

v8cffi.platform.**platform – Platform object (singleton)**

V8 platform environment. The underlying platform is a singleton that must only be initialized once per process.

Should be used through *platform*

Variables

- **natives_path** (*str*) – Path to natives_blob.bin
- **snapshot_path** (*str*) – Path to snapshot_blob.bin

2.1.2 VM Object

class v8cffi.vm.**VM**(*platform*)

Holds the VM state (V8 isolate). Running scripts within a VM is thread-safe, but only a single thread will execute code at a given time (there is a Global Lock). It's feasible to run one VM per thread or to have a pre-initialized pool.

There may be many VMs per platform

Parameters **platform** (*_Platform*) – Initialized platform

create_context()

Create a *Context* for running JS scripts

Returns Instance of *Context*

Return type *Context*

get_c_vm()

@Private Return the underlying C VM

Returns struct cdata

Return type ffi.CData

is_alive()

Check the vm is initialized and was not exited

Returns Whether the vm is alive or not

Return type bool

set_up()

Initialize the VM. Remember to call *tear_down()* before exiting the application. It's recommended to use a with statement instead of this method to ensure clean up

Raises **V8MemoryError** – if there is no memory for allocating it, the process should die afterwards anyway, there is little point in catching this

tear_down()

Destructs the VM

2.1.3 Context Object

class v8cffi.context.**Context**(*vm*)

An execution environment that allows separate, unrelated, JS applications to run in a single instance of V8. It may be thought as a browser tab.

Running scripts within the same Context is thread-safe.

There may be many Contexts per VM

Parameters **vm** (*VM*) – Initialized VM

load_libs(*scripts_paths*)
Load script files into the context. This can be thought as the HTML script tag. The files content must be utf-8 encoded.

This is a shortcut for reading the files and pass the content to *run_script()*

Parameters **scripts_paths** (*list*) – Script file paths.

Raises

- **OSError** – If there was an error manipulating the files. This should not normally be caught
- **V8Error** – if there was an error running the JS script

run_script(*script*, *identifier='<anonymous>'*)
Run a JS script within the context. All code is ran synchronously, there is no event loop. It's thread-safe

Parameters

- **script** (*bytes or str*) – utf-8 encoded or unicode string
- **identifier** (*bytes or str*) – utf-8 encoded or unicode string. This is used as the name of the script (ie: in stack-traces)

Returns Result of running the JS script

Return type str

Raises **V8Error** – if there was an error running the JS script

set_up()
Initialize the context. Remember to call *tear_down()* before exiting the application. It's recommended to use a `with` statement instead of this method to ensure clean up

Raises **V8MemoryError** – if there is no memory for allocating it, the process should die afterwards anyway, there is little point in catching this

tear_down()
Destructs the context

2.1.4 Shortcuts Module

v8cffi.shortcuts.**set_up**()
Set ups the V8 machinery: platform, VM and context.

This function is not thread-safe, it must be called from a place where is guaranteed it will be called once and only once. Probably within the main-thread at import time.

v8cffi.shortcuts.**get_context**()
Return a global V8 context.

set_up() must has been called

Returns Global V8 context

Return type *Context*

CHAPTER 3

Additional Notes

Design notes, legal information and changelog.

3.1 Distribution decisions

There are a couple of ways I could have distributed the library:

1. Distribute a `libv8cffi.so` or `libv8.so` as a separate package, install it into `LD_LIBRARY_PATH` and copy `natives_blob.bin` + `snapshot_blob.bin` into somewhere.
2. Compile everything from source at install time.

The second option is probably the worst since the V8 repo alone is about 800MB and it takes quite a few minutes to fetch and compile. The first option is ok I guess, but I'd rather prefer the user compiles the library them-self.

So, I went with some intermediate option: distribute the V8 static libraries (.a) when possible and compile the cffi wrapper at install time.

3.2 Changelog

3.2.1 0.2.1

- Python 2.7 support
- PyPy 5.3

3.2.2 0.2.0

- *Context.run_script* takes a new parameter *ìdentifier*
- JS error contains a trace back
- Linux binaries compiled in CentOS6
- V8 version: 4.9.385.33
- Update cffi dependency to 1.6

3.2.3 0.1.0

- Initial release

3.3 License

V

Symbols

C

G

I

L

P

R

S

T

V