Scell Documentation

Release 0.2.0

Eugene Eeo

May 14, 2015

Contents

Scell is a MIT-licensed selector library, written in Python.

At the heart of event-driven platforms is the selector, the humble **select.select** function. Using it is very simple, but requires you to pass in lists of file handles again and again. Keeping track of these file handles are tricky. Let Scell handle it for you and focus on building an awesome library/server:

```
>>> selector = scell.Selector()
>>> monitor = selector.register(open('file.txt'), mode='r')
>>> monitor.callback = lambda: 1
>>> [event.callback() for event in selector.ready]
[1]
```

Scell allows implementors to effortlessly build libraries atop of the minimal abstraction provided, while also having good performance. The API is very small and works across many platforms, including Windows and OSX.

CHAPTER 1

Contents

1.1 Guide

Scell is a very simple library that is comprised of two main components- the core library, which implements wrappers around monitored file objects and the `select.select()` function, and the wrapper which implements a *scell.wrapper.Selector* object atop a dictionary. To get started with the rest of the guide first create a *scell.wrapper.Selector* instance:

```
selector = Selector()
```

1.1.1 Registering File Objects

The recommended way to register file objects for their interests is using the *scell.wrapper.Selector.scoped()* context manager:

```
with selector.scoped([fp1, fp2]) as (m1, m2):
    pass
```

Within the body of the `with` block, the file objects are registered but once the block exits the selector automatically unregisters them. This ensures that file objects are automatically cleaned up and that unneeded resources can be freed. You can also use the alternative forms:

```
# more verbose but more control
from scell.core import Monitored
selector[fp] = Monitored(
    fp,
    wants_read=True,
    wants_write=True,
)

# easier but less control
selector.register(fp, mode='rw')
```

You can also attach callbacks or other form of data alongside the registered file object:

```
monitor = selector.register(fp, mode='rw')
monitor.callback = lambda x: x
```

1.1.2 Querying Events

You can query for readability or writability of each file object by simply using the *scell.wrapper.Selector.select()* method:

```
for event in selector.select():
    assert event.readable
    assert event.writable
```

The `select` method returns an iterable of *scell.core.Event* objects that represents the readability and writability of the monitored file objects. A code example of the attributes and what they are:

```
event.readable      # whether the file object is readable
event.writable      # whether the file object is writable
event.ready         # whether the monitored meets are met
event.fp            # underlying file object
event.callback      # callback associated with the file object
event.monitored     # monitored interests of file object
```

To only select file objects which are ready, use the *scell.wrapper.Selector.ready()* method. For example:

```
for event in selector.ready():
    assert event.ready
```

1.1.3 Cleaning Up

Cleaning up after ourselves is important- that is, to unregister file objects that have already been closed or unregister file objects that we are no longer interested in. If you used the *scell.wrapper.Selector.scoped()* you don't need to unregister any file objects.

To unregister file objects use the *scell.wrapper.Selector.unregister()* method:

```
selector.unregister(fp)
```

Note that it raises a `KeyError` if you unregister file objects that are not present.

1.2 API

1.2.1 scell.wrapper

Implements the `Selector` class, a high level wrapper around `~select.select`.

class `scell.wrapper.`**Selector**

A selector object is a dictionary of file-like objects to `Monitored` objects.

info()

D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

ready()

Yields the registered monitors which are ready (their interests are satisfied).

register(*fp*, *mode*)

Register a given *fp* (file handle) under a given *mode*. The *mode* can either be `r`, `w`, or both. Returns the monitor object.

Parameters

- **fp** – The file-like object.
- **mode** – Whether read and or write-ready events should be notified.

registered
D.iteritems() -> an iterator over the (key, value) items of D

rwlist()
Returns `(rl, wl)` where `rl` and `wl` are file objects interested in readability and writability, respectively.

scoped(**args*, ***kwds*)
A context manager that automatically unregisters **fps** once the block has finished executing.

Parameters

- **fps** – Iterable of file objects.
- **mode** – Defaults to 'rw', the interests of every file handle.

select(*timeout=None*)
Returns an iterable of monitors which are readable or writable subject to *timeout*.

Parameters **timeout** – Maximum number of seconds to wait until at least 1 file object is readable or writable. To block for an indefinite time, use `None`.

unregister
x.__delitem__(y) <==> del x[y]

1.2.2 scell.core

Provides abstractions over lower level APIs and file objects and their interests.

class `scell.core.`**Event**(*monitored*, *readable*, *writable*)
Represents the readability or writability of a *monitored* file object.

ready
Whether the *monitored* needs are met, i.e. whether it is readable or writable, taking it's needs into account.

class `scell.core.`**Monitored**(*fp*, *wants_read*, *wants_write*)
Represents the interests of a file handle *fp*, and whether it *wants_read* and or *wants_write*.

`scell.core.`**select**(*rl*, *wl*, *timeout=None*)
Returns the file objects ready for reading/writing from the read-list (*rl*) and write-list (*wl*), subject to *timeout* in seconds.

Parameters

- **rl** – Objects interested in readability.
- **wl** – Objects interested in writability.
- **timeout** – Maximum blocking time in seconds, *None* for no timeout.

S

E

I

M

R

S

U