liberator Documentation

Release 0.0.3

joncrall

Aug 22, 2023

CONTENTS

Github	https://gitlab.kitware.com/python/liberator
Pypi	https://pypi.org/project/liberator
ReadTheDocs	https://liberator.readthedocs.io/en/latest/

CHAPTER

ONE

LIBERATOR PACKAGE

1.1 Submodules

1.1.1 liberator.core module

Extracts relevant parts of the source code

Note: If the source code changes while the run is executing then this may not work correctly.

TODO:

- [x] Maintain a parse tree instead of raw lines
- [x] Keep a mapping from "definition names" to the top-level nodes in the parse tree that define them.
- [X] For each extracted node in the parse tree keep track of
  - [X] where it came from
  - [ ] what modifications were made to it
- [ ] Handle expanding imports nested within functions
- [ ] Maintain docstring formatting after using the node transformer

Issues:

- [ ] We currently (0.0.1) get a KeyError in the case where, a module is imported like *import mod.submod* and all usage is of the form *mod.submod.attr*, then

`class` liberator.core.**Liberator**(*tag='root'*, *logger=None*, *verbose=0*)

Bases: NiceRepr

Maintains the current state of the source code

There are 3 major steps:

(a) extract the code to that defines a function or class from a module,

(b) go back to the module and extract extra code required to define any names that were undefined in the extracted code, and

(c) replace import statements to specified "expand" modules with the actual code used to define the variables accessed via the imports.

This results in a standalone file that has absolutely no dependency on the original module or the specified "expand" modules (the expand module is usually the module that is doing the training for a network. This means that you can deploy a model independant of the training framework).

Note: This is not designed to work for cases where the code depends on logic executed in a global scope (e.g. dynamically registering properties) . I think its actually impossible to statically account for this case in general.

Parameters

- **tag** (*str*) – logging tag
- **logger** (*Callable*) – logging function
- **verbose** (*int*) – verbosity, 0 is nothing, 1 is info, 2 is debug, etc..

Example

```
>>> import ubelt as ub
>>> from liberator.core import Liberator
>>> lib = Liberator(logger=print)
>>> lib.add_dynamic(ub.find_exe, eager=False)
>>> lib.expand(['ubelt'])
>>> print(lib.current_sourcecode())
```

```
>>> lib = Liberator()
>>> lib.add_dynamic(ub.find_exe, eager=True)
>>> print(lib.current_sourcecode())
```

```
>>> lib = Liberator(logger=3, tag='mytest')
>>> lib.add_dynamic(ub.Cacher, eager=True)
>>> visitor = ub.peek(lib.visitors.values())
>>> print('visitor.definitions = {}'.format(ub.urepr(ub.map_keys(str, visitor.
↪definitions), nl=1)))
>>> print('visitor.nested_definitions = {}'.format(ub.urepr(ub.map_keys(str,␣
↪visitor.nested_definitions), nl=1)))
```

```
>>> lib._print_logs()
>>> lib.expand(['ubelt'])
```

Example

```
>>> # xdoctest: +REQUIRES(module:fastai)
>>> from liberator.core import *
>>> import fastai.vision
>>> obj = fastai.vision.models.WideResNet
>>> expand_names = ['fastai']
>>> lib = Liberator()
>>> lib.add_dynamic(obj)
>>> lib.expand(expand_names)
>>> #print(ub.urepr(lib.body_defs, si=1))
>>> print(lib.current_sourcecode())
```

Example

```
>>> # xdoctest: +REQUIRES(module:fastai)
>>> from liberator.core import Liberator
>>> from fastai.vision.models import unet
>>> lib = Liberator()
>>> lib.add_dynamic(unet.DynamicUnet)
>>> lib.expand(['fastai'])
>>> print(lib.current_sourcecode())
```

Example

```
>>> # xdoctest: +REQUIRES(module:netharn)
>>> from liberator.core import *
>>> import netharn as nh
>>> from netharn.models.yolo2 import yolo2
>>> obj = yolo2.Yolo2
>>> expand_names = ['netharn']
>>> lib = Liberator()
>>> lib.add_static(obj.__name__, sys.modules[obj.__module__].__file__)
>>> lib.expand(expand_names)
>>> #print(ub.urepr(lib.body_defs, si=1))
>>> print(lib.current_sourcecode())
```

error(*msg*)

info(*msg*)

debug(*msg*)

warn(*msg*)

_print_logs()

_add_definition(*d*)

current_sourcecode()

_ensure_visitor(*modpath=None*, *module=None*)

Return an existing visitor for a module or create one if it doesnt exist

add_dynamic(*obj*, *eager=True*)

Add the source to define a live python object

Parameters

- **obj** (*object*) – a reference to a class or function
- **eager** (*bool*) – experimental

Example

```
>>> from liberator import core
>>> import liberator
>>> obj = core.unparse
>>> eager = True
>>> lib = liberator.Liberator()
>>> lib.add_dynamic(obj, eager=eager)
>>> print(lib.current_sourcecode())
```

add_static(*name*, *modpath*)

Statically extract a definition from a module file

Parameters

- **name** (*str*) – the name of the member of the module to define
- **modpath** (*PathLike*) – The path to the module

Example

```
>>> from liberator import core
>>> import liberator
>>> modpath = core.__file__
>>> name = core.unparse.__name__
>>> lib = liberator.Liberator()
>>> lib.add_static(name, modpath)
>>> print(lib.current_sourcecode())
```

_lazy_close()

close2(*visitors*)

Experimental

Populate all undefined names using the context from a module

close(*visitor*)

Populate all undefined names using the context from a module

expand(*expand_names*)

Remove all references to specific modules by directly copying in the referenced source code. If the code is referenced from a module, then the references will need to change as well.

Parameters

expand_name (*List[str]*) – list of module names. For each module we expand any reference to that module in the closed source code by directly copying the referenced code into that file. This doesn't work in all cases, but it usually does. Reasons why this wouldn't work include trying to expand import from C-extension modules and expanding modules with complicated global-level logic.

Todo:

- **[ ] Add special unique (mangled) suffixes to all expanded names**
  to avoid name conflicts.

Example

```
>>> # Test a heavier duty class
>>> # xdoctest: +REQUIRES(module:netharn)
>>> from liberator.core import *
>>> import netharn as nh
>>> obj = nh.device.MountedModel
>>> #obj = nh.layers.ConvNormNd
>>> #obj = nh.data.CocoDataset
>>> #expand_names = ['ubelt', 'progiter']
>>> expand_names = ['netharn']
>>> lib = Liberator()
>>> lib.add_dynamic(obj)
>>> lib.expand(expand_names)
>>> #print('header_defs = ' + ub.urepr(lib.header_defs, si=1))
>>> #print('body_defs = ' + ub.urepr(lib.body_defs, si=1))
>>> print('SOURCE:')
>>> text = lib.current_sourcecode()
>>> print(text)
```

expand_module_attributes(*d*)

Parameters
d (*Definition*) – the definition to expand

class liberator.core.**Closer**(*tag='root'*, *logger=None*, *verbose=0*)

Bases: *Liberator*

Deprecated in favor of *Liberator*.

The original name of the Liberator class was called Closer. Exposing this for backwards compatibility.

1.1.2 liberator.starfinder module

liberator.starfinder.**find_import_stars**(*text*)

Parameters
text (*str*) – the python code to refactor

Example

```
>>> # xdoctest: +REQUIRES(module:parso)
>>> from liberator.starfinder import *  # NOQA
>>> import ubelt as ub
>>> text = ub.codeblock(
>>>     '''
>>>     import dis as dat
>>>     from io import *
>>>     from a.b import *
>>>     from textwrap import *  # NOQA
>>>     x = StringIO
>>>     y = dedent
>>>     ''')
```

(continues on next page)

(continued from previous page)

```
>>> final_text = find_import_stars(text)
>>> print('----')
>>> print('Text')
>>> print('----')
>>> print(ub.highlight_code(text))
>>> print('----------')
>>> print('Final Text')
>>> print('----------')
>>> print(ub.highlight_code(final_text))
```

1.2 Module contents

Github	https://gitlab.kitware.com/python/liberator
Pypi	https://pypi.org/project/liberator
ReadTheDocs	https://liberator.readthedocs.io/en/latest/

class liberator.**Closer**(*tag='root'*, *logger=None*, *verbose=0*)

Bases: *Liberator*

Deprecated in favor of *Liberator*.

The original name of the Liberator class was called Closer. Exposing this for backwards compatibility.

class liberator.**Liberator**(*tag='root'*, *logger=None*, *verbose=0*)

Bases: NiceRepr

Maintains the current state of the source code

There are 3 major steps:

(a) extract the code to that defines a function or class from a module,

(b) go back to the module and extract extra code required to define any names that were undefined in the extracted code, and

(c) replace import statements to specified "expand" modules with the actual code used to define the variables accessed via the imports.

This results in a standalone file that has absolutely no dependency on the original module or the specified "expand" modules (the expand module is usually the module that is doing the training for a network. This means that you can deploy a model independant of the training framework).

Note: This is not designed to work for cases where the code depends on logic executed in a global scope (e.g. dynamically registering properties) . I think its actually impossible to statically account for this case in general.

Parameters

- **tag** (*str*) – logging tag
- **logger** (*Callable*) – logging function
- **verbose** (*int*) – verbosity, 0 is nothing, 1 is info, 2 is debug, etc..

Example

```
>>> import ubelt as ub
>>> from liberator.core import Liberator
>>> lib = Liberator(logger=print)
>>> lib.add_dynamic(ub.find_exe, eager=False)
>>> lib.expand(['ubelt'])
>>> print(lib.current_sourcecode())
```

```
>>> lib = Liberator()
>>> lib.add_dynamic(ub.find_exe, eager=True)
>>> print(lib.current_sourcecode())
```

```
>>> lib = Liberator(logger=3, tag='mytest')
>>> lib.add_dynamic(ub.Cacher, eager=True)
>>> visitor = ub.peek(lib.visitors.values())
>>> print('visitor.definitions = {}'.format(ub.urepr(ub.map_keys(str, visitor.
↪definitions), nl=1)))
>>> print('visitor.nested_definitions = {}'.format(ub.urepr(ub.map_keys(str,␣
↪visitor.nested_definitions), nl=1)))
```

```
>>> lib._print_logs()
>>> lib.expand(['ubelt'])
```

Example

```
>>> # xdoctest: +REQUIRES(module:fastai)
>>> from liberator.core import *
>>> import fastai.vision
>>> obj = fastai.vision.models.WideResNet
>>> expand_names = ['fastai']
>>> lib = Liberator()
>>> lib.add_dynamic(obj)
>>> lib.expand(expand_names)
>>> #print(ub.urepr(lib.body_defs, si=1))
>>> print(lib.current_sourcecode())
```

Example

```
>>> # xdoctest: +REQUIRES(module:fastai)
>>> from liberator.core import Liberator
>>> from fastai.vision.models import unet
>>> lib = Liberator()
>>> lib.add_dynamic(unet.DynamicUnet)
>>> lib.expand(['fastai'])
>>> print(lib.current_sourcecode())
```

Example

```
>>> # xdoctest: +REQUIRES(module:netharn)
>>> from liberator.core import *
>>> import netharn as nh
>>> from netharn.models.yolo2 import yolo2
>>> obj = yolo2.Yolo2
>>> expand_names = ['netharn']
>>> lib = Liberator()
>>> lib.add_static(obj.__name__, sys.modules[obj.__module__].__file__)
>>> lib.expand(expand_names)
>>> #print(ub.urepr(lib.body_defs, si=1))
>>> print(lib.current_sourcecode())
```

error(*msg*)

info(*msg*)

debug(*msg*)

warn(*msg*)

_print_logs()

_add_definition(*d*)

current_sourcecode()

_ensure_visitor(*modpath=None*, *module=None*)

Return an existing visitor for a module or create one if it doesnt exist

add_dynamic(*obj*, *eager=True*)

Add the source to define a live python object

Parameters

- **obj** (*object*) – a reference to a class or function
- **eager** (*bool*) – experimental

Example

```
>>> from liberator import core
>>> import liberator
>>> obj = core.unparse
>>> eager = True
>>> lib = liberator.Liberator()
>>> lib.add_dynamic(obj, eager=eager)
>>> print(lib.current_sourcecode())
```

add_static(*name*, *modpath*)

Statically extract a definition from a module file

Parameters

- **name** (*str*) – the name of the member of the module to define
- **modpath** (*PathLike*) – The path to the module

Example

```
>>> from liberator import core
>>> import liberator
>>> modpath = core.__file__
>>> name = core.unparse.__name__
>>> lib = liberator.Liberator()
>>> lib.add_static(name, modpath)
>>> print(lib.current_sourcecode())
```

_lazy_close()

close2(*visitors*)

Experimental

Populate all undefined names using the context from a module

close(*visitor*)

Populate all undefined names using the context from a module

expand(*expand_names*)

Remove all references to specific modules by directly copying in the referenced source code. If the code is referenced from a module, then the references will need to change as well.

Parameters

expand_name (*List[str]*) – list of module names. For each module we expand any reference to that module in the closed source code by directly copying the referenced code into that file. This doesn't work in all cases, but it usually does. Reasons why this wouldn't work include trying to expand import from C-extension modules and expanding modules with complicated global-level logic.

Todo:

- **[ ] Add special unique (mangled) suffixes to all expanded names**
  to avoid name conflicts.

Example

```
>>> # Test a heavier duty class
>>> # xdoctest: +REQUIRES(module:netharn)
>>> from liberator.core import *
>>> import netharn as nh
>>> obj = nh.device.MountedModel
>>> #obj = nh.layers.ConvNormNd
>>> #obj = nh.data.CocoDataset
>>> #expand_names = ['ubelt', 'progiter']
>>> expand_names = ['netharn']
>>> lib = Liberator()
>>> lib.add_dynamic(obj)
>>> lib.expand(expand_names)
>>> #print('header_defs = ' + ub.urepr(lib.header_defs, si=1))
>>> #print('body_defs = ' + ub.urepr(lib.body_defs, si=1))
```

(continues on next page)

(continued from previous page)

```
>>> print('SOURCE:')
>>> text = lib.current_sourcecode()
>>> print(text)
```

expand_module_attributes(*d*)

Parameters
d (*Definition*) – the definition to expand

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

E

F

I

L

M

W