Dirty Loader Documentation

Release 0.0.1

alfred82santa

October 10, 2016

Contents

CHAPTER 1

Dirty Loader package

1.1 Loaders

class dirty_loader.**Loader**(*modules=None*, *factories=None*)
Bases: object

Loader is a class loader. You must register python modules where to look for classes. First modules registered has preference in front last ones, but you could indicate index where you want insert new module.

register_module(*module*, *idx=-1*)
Register a module. You could indicate position inside inner list.

Parameters

- **module** (*str*) – must be a string or a module object to register.
- **idx** (*int*) – position where you want to insert new module. By default it is inserted at the end.

unregister_module(*module*)
Unregister a module.

Parameters **module** (*str*) – must be a string or a module object to unregistered

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

load_class(*classname*)
Loads a class looking for it in each module registered.

Parameters **classname** (*str*) – Class name you want to load.

Returns Class object

Return type type

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_factory_by_class(*klass*)
Returns a custom factory for class. By default it will return the class itself.

Parameters **klass** (*type*) – Class type

Returns Class factory

Return type callable

class dirty_loader.**LoaderReversed**(*modules=None*, *factories=None*)
Bases: dirty_loader.ReversedMixin, *dirty_loader.Loader*

LoaderReversed is a class loader. You must register python modules where to look for classes. Last modules registered has preference in front first ones, but you could indicate index where you want insert new module.

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_factory_by_class(*klass*)
Returns a custom factory for class. By default it will return the class itself.

Parameters **klass** (*type*) – Class type

Returns Class factory

Return type callable

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

load_class(*classname*)
Loads a class looking for it in each module registered.

Parameters **classname** (*str*) – Class name you want to load.

Returns Class object

Return type type

register_module(*module*, *idx=-1*)
Register a module. You could indicate position inside inner list.

Parameters

- **module** (*str*) – must be a string or a module object to register.
- **idx** (*int*) – position where you want to insert new module. By default it is inserted at the end.

unregister_module(*module*)
Unregister a module.

Parameters **module** (*str*) – must be a string or a module object to unregistered

class dirty_loader.**LoaderCached**(**args*, ***kwargs*)
Bases: dirty_loader.CacheLoaderMixin, *dirty_loader.Loader*

LoaderCached is a class loader. You must register python modules where to look for classes. First modules registered has preference in front last ones, but you could indicate index where you want insert new module.

Already looked up classes are cached.

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

invalidate_cache()
Invalidate class cache.

invalidate_cache_factories()
Invalidate factories cache.

class dirty_loader.**LoaderReversedCached**(**args*, ***kwargs*)
Bases: dirty_loader.CacheLoaderMixin, *dirty_loader.LoaderReversed*

Already looked up classes are cached.LoaderReversedCached is a class loader. You must register python modules where to look for classes. Last modules registered has preference in front first ones, but you could indicate index where you want insert new module.

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

invalidate_cache()
Invalidate class cache.

invalidate_cache_factories()
Invalidate factories cache.

class dirty_loader.**LoaderNamespace**(*namespaces=None*, *factories=None*)
Bases: *dirty_loader.Loader*

LoaderNamespace is a class loader. You must register python modules with a namespace tag where to look for classes. First namespace registered has preference in front last ones.

register_module(*module*, *namespace=None*)
Register a module.

Parameters

- **module** (*str*) – must be a string or a module object to register.
- **namespace** (*str*) – Namespace tag. If it is None module will be used as namespace tag

register_namespace(*namespace*, *module*)
Register a namespace.

Parameters

- **namespace** (*str*) – Namespace tag.
- **module** (*str*) – must be a string or a module object to register.

unregister_module(*module*)
Unregister a module.

Parameters **module** (*str*) – must be a string or a module object to unregistered

unregister_namespace(*namespace*)
Unregister a namespace.

Parameters **namespace** (*str*) – Namespace tag.

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

get_registered_namespaces()
Return registered namespaces.

Returns Dict with namespaces as key and modules as value.

Return type OrderedDict

load_class(*classname*, *namespace=None*)
Loads a class looking for it in each module registered. It's possible to load a class from specific namespace using **namespace** parameter or using classname as "namespace:classname".

Parameters

- **classname** (*str*) – Class name you want to load.
- **namespace** (*str*) – Specific namespace where to look for class.

Returns Class object

Return type type

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_factory_by_class(*klass*)
Returns a custom factory for class. By default it will return the class itself.

Parameters **klass** (*type*) – Class type

Returns Class factory

Return type callable

class dirty_loader.**LoaderNamespaceReversed**(*namespaces=None*, *factories=None*)
Bases: dirty_loader.ReversedMixin, *dirty_loader.LoaderNamespace*

LoaderNamespaceReversed is a class loader. You must register python modules with a namespace tag where to look for classes. Last namespaces registered has preference in front first ones.

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_factory_by_class(*klass*)
Returns a custom factory for class. By default it will return the class itself.

Parameters **klass** (*type*) – Class type

Returns Class factory

Return type callable

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

get_registered_namespaces()
Return registered namespaces.

Returns Dict with namespaces as key and modules as value.

Return type OrderedDict

load_class(*classname*, *namespace=None*)
Loads a class looking for it in each module registered. It's possible to load a class from specific namespace using **namespace** parameter or using classname as "namespace:classname".

Parameters

- **classname** (*str*) – Class name you want to load.
- **namespace** (*str*) – Specific namespace where to look for class.

Returns Class object

Return type type

register_module(*module*, *namespace=None*)
Register a module.

Parameters

- **module** (*str*) – must be a string or a module object to register.
- **namespace** (*str*) – Namespace tag. If it is None module will be used as namespace tag

register_namespace(*namespace*, *module*)
Register a namespace.

Parameters

- **namespace** (*str*) – Namespace tag.
- **module** (*str*) – must be a string or a module object to register.

unregister_module(*module*)
Unregister a module.

Parameters **module** (*str*) – must be a string or a module object to unregistered

unregister_namespace(*namespace*)
Unregister a namespace.

Parameters **namespace** (*str*) – Namespace tag.

class dirty_loader.**LoaderNamespaceCached**(**args*, ***kwargs*)
Bases: dirty_loader.CacheLoaderNamespaceMixin, *dirty_loader.LoaderNamespace*

LoaderNamespace is a class loader. You must register python modules with a namespace tag where to look for classes. First namespace registered has preference in front last ones.

Already looked up classes are cached.

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

get_registered_namespaces()
Return registered namespaces.

Returns Dict with namespaces as key and modules as value.

Return type OrderedDict

invalidate_cache()
Invalidate class cache.

invalidate_cache_factories()
Invalidate factories cache.

class dirty_loader.**LoaderNamespaceReversedCached**(**args*, ***kwargs*)
Bases: dirty_loader.CacheLoaderNamespaceMixin, *dirty_loader.LoaderNamespaceReversed*

LoaderNamespaceReversed is a class loader. You must register python modules with a namespace tag where to look for classes. Last namespaces registered has preference in front first ones.

Already looked up classes are cached.

factory(*classname*, **args*, ***kwargs*)
Creates an instance of class looking for it in each module registered. You can add needed params to instance the class.

Parameters **classname** (*str*) – Class name you want to create an instance.

Returns An instance of classname

Return type object

get_registered_modules()
Return registered modules.

Returns list of registered modules.

Return type list

get_registered_namespaces()
Return registered namespaces.

Returns Dict with namespaces as key and modules as value.

Return type OrderedDict

invalidate_cache()
Invalidate class cache.

invalidate_cache_factories()
Invalidate factories cache.

dirty_loader.**import_class**(*classpath*, *package=None*)
Load and return a class

1.2 Factories

class dirty_loader.factories.**BaseFactory**(*loader*, *klass*)
Bases: object

Base class factory. It should be used in order to implement specific ones.

class dirty_loader.factories.**LoggerFactory**(*loader*, *klass*)
Bases: dirty_loader.factories.BaseLoggingFactory

Logger factory.

class dirty_loader.factories.**LoggingHandlerFactory**(*loader*, *klass*)
Bases: dirty_loader.factories.BaseLoggingFactory

Logger handle factory.

dirty_loader.factories.**register_logging_factories**(*loader*)
Registers default factories for logging standard package.

Parameters **loader** – Loader where you want register default logging factories

CHAPTER 2

Introduction

CHAPTER 3

dirty-loader

Easy to use loader library.

3.1 Changelog

3.1.1 Version 0.2.2

- Simplified code.
- Added BaseFactory methods to load items from list or dictionaries.

3.1.2 Version 0.2.1

- Three types of instance definition allowed: string, structured and structure simplified.

3.1.3 Version 0.2.0

- Custom factories for classes.
- Default factories for logging package.

3.1.4 Version 0.1.0

- Some refactors.
- New function `import_class`.

3.2 Instalation

```
$ pip install dirty-loader
```

3.3 Documentation

http://dirty-loader.readthedocs.io

3.4 Main loaders

3.4.1 Loader

With Loader you could register sorted python modules. When you ask for a class it will try to load it for each module until it find one.

Example:

```
from dirty_loader import Loader

loader = Loader()
loader.register_module('tests.fake.namespace1')
loader.register_module('tests.fake.namespace2')
loader.register_module('tests.fake.namespace3')

klass = loader.load_class('FakeClass1')

from tests.fake.namespace1 import FakeClass1
assert klass == FakeClass1

# klass is tests.fake.namespace1.FakeClass1 because it exists in first module registered.
# Also, you could get an instance of class using factory
obj = loader.factory('FakeClass1', var1='a', var2=2)

# You could load classes from packages inside modules registered
klass = loader.load_class('subnamespace.FakeClass1')
from tests.fake.namespace3.subnamespace import FakeClass1 as SubFakeClass1
assert klass == SubFakeClass1
# klass is tests.fake.namespace3.subnamespace.FakeClass1 because it exists in first module registered
```

3.4.2 LoaderReversed

It works in same way of Loader but it revers the sort when try to load a class.

Example:

```
from dirty_loader import LoaderReversed

loader = LoaderReversed()
loader.register_module('tests.fake.namespace1')
loader.register_module('tests.fake.namespace2')

klass = loader.load_class('FakeClass1')

from tests.fake.namespace2 import FakeClass1
assert klass == FakeClass1

# klass is tests.fake.namespace2.FakeClass1 because it exists in last module registered.
```

3.4.3 LoaderNamespace

With LoaderNamespace you could register sorted namespaces. When you ask for a class it will try to load it for each namespace until it find one. Each namespace has a python module associated. You could use the regular Loader way to load a class or you could specify the namespace you would like to use.

Example:

```
from dirty_loader import LoaderNamespace

loader = LoaderNamespace()
loader.register_namespace('fake1', 'tests.fake.namespace1')
loader.register_namespace('fake2', 'tests.fake.namespace2')

from tests.fake.namespace1 import FakeClass1, FakeClass2, FakeClass3

klass = loader.load_class('FakeClass1')

from tests.fake.namespace1 import FakeClass1
assert klass == FakeClass1
# klass is tests.fake.namespace1.FakeClass1 because it exists in last module registered.

# Also, you could get a class from specific namespace

klass = loader.load_class('FakeClass1', namespace='fake2)

from tests.fake.namespace2 import FakeClass1
assert klass == FakeClass1
# klass is tests.fake.namespace2.FakeClass1 because you specified it.

# Namespace could be specified in string class, too
klass = loader.load_class('fake2:FakeClass1')

assert klass == FakeClass1
# klass is tests.fake.namespace2.FakeClass1 because you specified it.
```

3.4.4 LoaderNamespaceReversed

It works in same way of LoaderNamespace but it revers the sort when try to load a class.

3.4.5 LoaderCached

A version of Loader with cache.

3.4.6 LoaderReversedCached

A version of LoaderReversed with cache.

3.4.7 LoaderNamespaceCached

A version of LoaderNamespace with cache.

3.4.8 LoaderNamespaceReversedCached

A version of LoaderNamespaceReversed with cache.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

d

B

D

F

G

I

L

R

U