Pynsive Documentation

Release 0.2.3

John Hopper

December 13, 2013

Contents

Pynsive is a simple plugin library that uses the **sys.meta_path** list along with custom finder and loader definitions to hook into the Python import process. This means that when directores or other plugin search targets are added in Pynsive, future import statements will now search the newly added path for plugin modules and classes.

For more information on the import process hooks, please see:

- Python 3 Import Process
- PEP-302

CHAPTER 1

Getting Started

Below are some helpful documents to help get you started in using Pynsive.

- [Blog Post] Getting started with Pynsive by Chad Lung

CHAPTER 2

Pynsive Documentation

2.1 Using Pynsive

2.1.1 Creating and Using a Plugin Context

The plugin context is a nice way of managing what directories you've plugged into the **sys.meta_path** variable. Managers may be destroyed when no longer needed. Destroying a manager removes all directories that the manager plugged into from the **sys.meta_path** variable.

```
import pynsive

plugin_manager = pynsive.PluginManager()
plugin_manager.plug_into('/some/path')

try:
    import myplugins.module.plugin_a as plugin
    print('Imported plugin module: {1}', plugin)
finally:
    plugin_manager.destroy()
```

2.1.2 Discovering Python Modules

Pynsive allows you to search a given directory tree for potential module names to aid in discovery of interesting code. These functions will search any directories found under the directory specified for python modules and will recurse as specified by their names.

```
import pynsive

# Non-recursive search
found_modules = pynsive.discover_modules('/some/path')
print('Discovered {1} modules.', len(found_modules))

# Recursive search
found_modules = pynsive.rdiscover_modules('/some/path')
print('Discovered {1} modules.', len(found_modules))
```

2.1.3 Dynamically Listing Submodules

Note: The list functions in Pynsive **will not** descend into the submodules that may exist under the specified module. In order to recursively search use the rlist functions.

```
import pynsive
import test_module

plugin_manager = pynsive.PluginManager()
plugin_manager.plug_into('/some/path')

try:
    found_modules = pynsive.list_modules('ext.plugins')
    print('Discovered {1} modules.', len(found_modules))
finally:
    plugin_manager.destroy()
```

2.1.4 Dynamically Finding Classes in a Module

Note: The list functions in Pynsive **will not** descend into the submodules that may exist under the specified module. In order to recursively search use the rlist functions.

```
import pynsive
import test_module

plugin_manager = pynsive.PluginManager()
plugin_manager.plug_into('/some/path')

try:
    def subclasses_only(type_to_test):
        same = type_to_test is not test_module.MyClass
        is_subclass = issubclass(type_to_test, test_module.MyClass)
        return not same and is_subclass

    classes = pynsive.list_classes('ext.plugins', subclasses_only)
    print('Discovered {1} classes.', len(classes))
finally:
    plugin_manager.destroy()
```

2.1.5 Dynamically Finding Classes in a Module and its Submodules

Note: The rlist functions in Pynsive **will** descend into the submodules that may exist under the specified module. In order to perform a non-recursive listing use the list functions.

```
import pynsive
import test_module

plugin_manager = pynsive.PluginManager()
plugin_manager.plug_into('/some/path')

try:
    def subclasses_only(type_to_test):
        same = type_to_test is not test_module.MyClass
        is_subclass = issubclass(type_to_test, test_module.MyClass)
        return not same and is_subclass
```

```
    classes = pynsive.rlist_classes('ext.plugins', subclasses_only)
    print('Discovered {1} classes.', len(classes))
finally:
    plugin_manager.destroy()
```

2.2 Pynsive API Documentation

2.2.1 `pynsive` Module

2.2.2 `common` Module

pynsive.common.**import_module**(*module_name*)
A nice namespace alias for users.

2.2.3 `reflection` Module

pynsive.reflection.**discover_modules**(*directory*)
Attempts to list all of the modules and submodules found within a given directory tree. This function searches the top-level of the directory tree for potential python modules and returns a list of candidate names.

Note: This function returns a list of strings representing discovered module names, not the actual, loaded modules.

Parameters **directory** – the directory to search for modules.

pynsive.reflection.**list_classes**(*mname*, *cls_filter=None*)
Attempts to list all of the classes within a specified module. This function works for modules located in the default path as well as extended paths via the sys.meta_path hooks.

If a class filter is set, it will be called with each class as its parameter. This filter's return value must be interpretable as a boolean. Results that evaluate as True will include the type in the list of returned classes. Results that evaluate as False will exclude the type in the list of returned classes.

Parameters

- **mname** – of the module to descend into
- **cls_filter** – a function to call to determine what classes should be included.

pynsive.reflection.**list_modules**(*mname*)
Attempts to list all of the submodules under a module. This function works for modules located in the default path as well as extended paths via the sys.meta_path hooks.

This function carries the expectation that the hidden module variable '__path__' has been set correctly.

Parameters **mname** – the module name to descend into

pynsive.reflection.**rdiscover_modules**(*directory*)
Attempts to list all of the modules and submodules found within a given directory tree. This function recursively searches the directory tree for potential python modules and returns a list of candidate names.

Note: This function returns a list of strings representing discovered module names, not the actual, loaded modules.

Parameters **directory** – the directory to search for modules.

pynsive.reflection.**rlist_classes**(*module*, *cls_filter=None*)
Attempts to list all of the classes within a given module namespace. This method, unlike list_classes, will recurse into discovered submodules.

If a type filter is set, it will be called with each class as its parameter. This filter's return value must be interpretable as a boolean. Results that evaluate as True will include the type in the list of returned classes. Results that evaluate as False will exclude the type in the list of returned classes.

Parameters

- **mname** – of the module to descend into
- **cls_filter** – a function to call to determine what classes should be included.

pynsive.reflection.**rlist_modules**(*mname*)
Attempts to the submodules under a module recursively. This function works for modules located in the default path as well as extended paths via the sys.meta_path hooks.

This function carries the expectation that the hidden module variable '__path__' has been set correctly.

Parameters **mname** – the module name to descend into

2.2.4 Subpackages

plugin Package

plugin Module

loader Module

exception pynsive.plugin.loader.**LoaderError**(*msg*)
Bases: exceptions.ImportError

This error is thrown when the module loader encounters an exception or an unrecoverable state while attempting to load a dynamically located module.

class pynsive.plugin.loader.**ModuleFinder**(*paths=None*)
Bases: object

As per PEP302, this module loader provides all of the necessary context for dynamically locating python modules based. This finder searches directories based on paths added to its internal list of available search directories.

add_path(*path*)
Adds a path to search through when attempting to look up a module.

Parameters **path** – the path the add to the list of searchable paths

find_module(*module_name*, *path=None*)
Searches the paths for the required module.

Parameters

- **module_name** – the full name of the module to find
- **path** – set to None when the module in being searched for is a top-level module - otherwise this is set to package.__path__ for submodules and subpackages (unused)

class pynsive.plugin.loader.**ModuleLoader**(*module_path*, *module_name*, *load_target*, *is_pkg*)
Bases: object

As per PEP302, this module loader provides all of the necessary context for loading a python file and executing its contents.

load_module(*module_name*)
Loads a module's code and sets the module's expected hidden variables. For more information on these variables and what they are for, please see PEP302.

Parameters **module_name** – the full name of the module to load

manager Module

class pynsive.plugin.manager.**PluginManager**
Bases: object

The PluginManager class is a nice wrapper for tapping into sys.meta_path with pynsive. It also provides a hook for removing itself from sys.meta_path.

destroy()
Removes the meta_path hook associated with this manager.

plug_into(**paths*)
Adds all arguments passed as plugin directories to search when loading modules.

2.2.5 Indices

- *genindex*
- *modindex*

2.3 Pynsive Changelog

2.3.1 Version 0.2.3

Module searching features exposed.

- **discover_modules** and **rdiscover_modules** added to allow for the recursive discovery of python modules within a given path.

2.3.2 Version 0.2.2

Minor bugfixes related to the recursive discovery of modules.

2.3.3 Version 0.2.1

Documentation changes, updates and a license swap.

- Lots of documentation updates
- Changed to MIT license

2.3.4 Version 0.2.0

Refactoring and cleanup.

- Fixed a bug where system paths weren't being correctly considering during recursive module descent operations

2.3.5 Version 0.1.4

This version has the following changes from release versions <= 0.1.4

- **discover_classes** was renamed to **rlist_classes**
- **discover_modules** was renamed to **list_modules**

CHAPTER 3

That Legal Thing...

Python Module Index

p