importscan Documentation

Release 0.2.dev0

Martijn Faassen

March 30, 2016

Contents

1 CHANGES **3**
- 1.1 0.2 (unreleased) . . . 3
- 1.2 0.1 (2016-03-15) . . . 3

2 Indices and tables **5**

Python Module Index **7**

importscan.**scan**(*package*, *ignore=None*, *handle_error=None*)

Scan a package by importing it.

A framework can provide registration decorators: a decorator that when used on a class or a function causes it to be registered with the framework. Metaclasses can also be used for this effect. As a consequence of this, registration only takes place when the module is actually imported. You can do this import explicitly in your code. It can however also be convenient to import everything in a package all at once automatically. This is what `scan` does.

This function was extracted and refactored from the `Venusian` library which also provides infrastructure for finding such decorators.

Parameters

- **package** – A reference to a Python package or module object.
- **ignore** – Ignore certain modules or packages during a scan. It should be a sequence containing strings and/or callables that are used to match against the dotted name of a module encountered during a scan. The sequence can contain any of these three types of objects:
  - A string representing a dotted name. For example, if you want to ignore the `my.package` package *and any of its submodules* during the scan, pass `ignore=['my.package']`.
  - A string representing a relative dotted name, a string starting with a dot. The relative module or package is relative to the `package` being scanned, so this does *not* match deeper relative packages.

    For example, if the `package` you've passed is imported as `my.package`, and you pass `ignore=['.mymodule']`, the `my.package.mymodule` mymodule *and any of its submodules* are omitted during scan processing. But `my.package.sub.mymodule` is *not* ignored as `mymodule` is nested in `sub`.
  - A callable that accepts a dotted name indicating a module or a package as its single positional argument and returns `True` or `False`. For example, if you want to skip all packages and modules with a full dotted path that ends with the word "tests", you can use `ignore=[re.compile('tests$').search]`. If the callable returns `True` (or anything else truthy), the object is ignored, if it returns `False` (or anything else falsy) the object is not ignored.

  You can mix and match the three types of strings in the list. For example, if the package being scanned is `my`, `ignore=['my.package', '.someothermodule', re.compile('tests$').search]` would cause `my.package` (and all its submodules and subobjects) to be ignored, `my.someothermodule` to be ignored, and any modules, packages, or global objects found during the scan that have a full dotted name that ends with the word `tests` to be ignored.

  Packages and modules matched by any ignore in the list are not imported, and their top-level (registration) code is not run as a result.

  You can also pass in a string or callable by itself as a single non-list argument.
- **handle_error** – A callback function that is called when an exception is raised during the importing process. By default `scan` propagates all errors that happen during the import process, including `ImportError`. If you use a custom `handle_error` callback, you can change this behavior by not reraising the error.

  Here's an example `handle_error` callback that ignores `ImportError` but not any other errors:

```
def handle_error(name, e):
    if not isinstance(e, ImportError):
        raise e
```

The first argument passed to `handle_error` is the module or package dotted name that could not be imported due to an exception. The second argument is the exception object. If `handle_error` does not re-raise the error, the error is suppressed.

CHAPTER 1

CHANGES

1.1 0.2 (unreleased)

- Nothing changed yet.

1.2 0.1 (2016-03-15)

- Initial public release.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

i

importscan, 3

I

importscan (module), 1

S

scan() (in module importscan), 1