Hacking The Python Import System

Liad Oz
https://github.com/LiadOz/

Terms

From python glossary:

- `Module` - An object that serves as an organizational unit of Python code
- `Package` - A Python module which can contain submodules or recursively, subpackages
- `Loader` - An object that loads a module
- `Finder` - An object that tries to find the loader for a module that is being imported

Import Machinery

Finder

```python
class ExampleFinder(importlib.abc.MetaPathFinder):
    def find_spec(self, fullname, path, target = None) -> Optional[ModuleSpec]:
        ...
```

- `fullname` - The full name of the module imported
- `path` - Path of the package the module is imported from
- `return value` - An object that contains the `Loader` that should be used to load the module

Finder

```python
class ExampleFinder(importlib.abc.MetaPathFinder):
    def find_spec(self, fullname, path, target = None) -> Optional[ModuleSpec]:
        ...
```

- `fullname` - The full name of the module imported
- `path` - Path of the package the module is imported from
- `return value` - An object that contains the `Loader` that should be used to load the module

Finder

```
class ExampleFinder(importlib.abc.MetaPathFinder):
    def find_spec(self, fullname, path, target = None) -> Optional[ModuleSpec]:
        ...
```

- `fullname` - The full name of the module imported
- `path` - Path of the package the module is imported from
- `return value` - An object that contains the `Loader` that should be used to load the module

Example:

```
from flask.json import loads
```

Finder

```
class ExampleFinder(importlib.abc.MetaPathFinder):
    def find_spec(self, fullname, path, target = None) -> Optional[ModuleSpec]:
        ...
```

- `fullname` - The full name of the module imported
- `path` - Path of the package the module is imported from
- `return value` - An object that contains the `Loader` that should be used to load the module

Example:

```
from flask.json import loads
```

- `fullname`: `"flask.json"`
- `path`: `["/Users/loz/env/lib/python3.9/site-packages/flask"]`

Loader

```python
class ExampleLoader(importlib.abc.Loader):
    def create_module(self, spec: ModuleSpec) -> ModuleType:
        ...

    def exec_module(self, module: ModuleType) -> None:
        ...
```

- `spec` - The module spec created by `find_spec`
- `module` - The module object created by `create_module`

Loader

```python
class ExampleLoader(importlib.abc.Loader):
    def create_module(self, spec: ModuleSpec) -> ModuleType:
        ...

    def exec_module(self, module: ModuleType) -> None:
        ...
```

- `spec` - The module spec created by `find_spec`
- `module` - The module object created by `create_module`

Loader

```python
class ExampleLoader(importlib.abc.Loader):
    def create_module(self, spec: ModuleSpec) -> ModuleType:
        ...

    def exec_module(self, module: ModuleType) -> None:
        ...
```

- `spec` - The module spec created by `find_spec`
- `module` - The module object created by `create_module`

The result of the `create_module` is the module returned from the import statement

sys.meta_path

A list of all the Finders that are used in the Import Machinery

```
import sys
sys.meta_path
```

sys.meta_path

A list of all the Finders that are used in the Import Machinery

```
import sys
sys.meta_path
```

```
[<class '_frozen_importlib.BuiltinImporter'>,
 <class '_frozen_importlib.FrozenImporter'>,
 <class '_frozen_importlib_external.PathFinder'>]
```

sys.meta_path

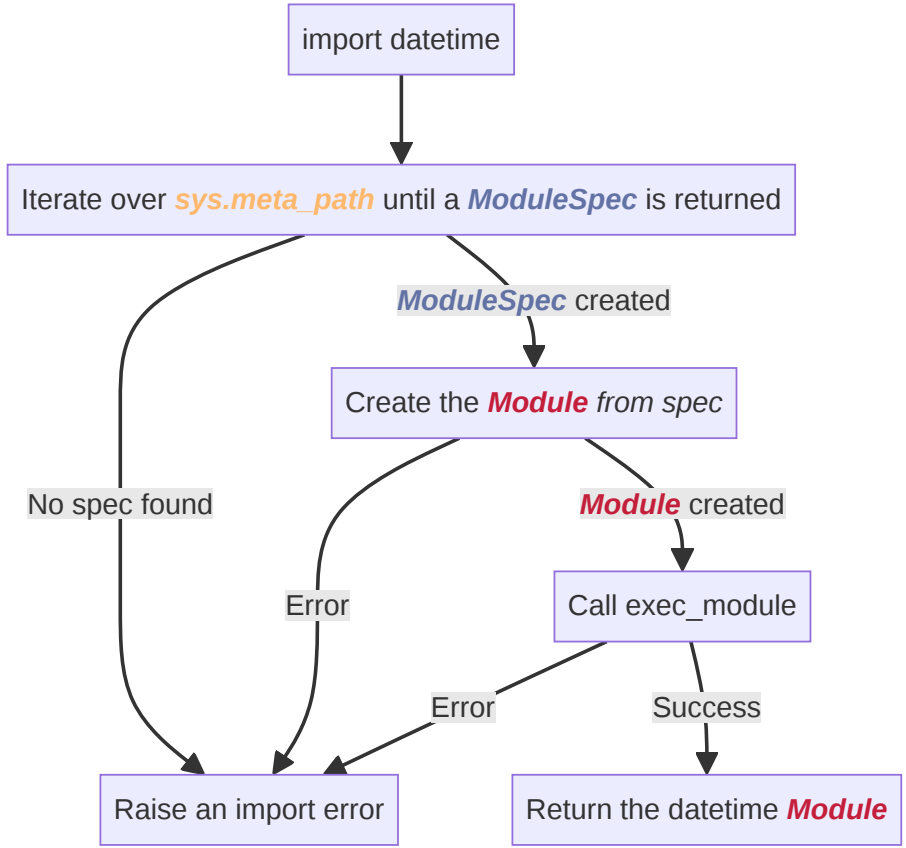
A list of all the Finders that are used in the Import Machinery

```
import sys
sys.meta_path
```

```
[<class '_frozen_importlib.BuiltinImporter'>,
 <class '_frozen_importlib.FrozenImporter'>,
 <class '_frozen_importlib_external.PathFinder'>]
```

- `BuiltinImporter` - For modules like `sys` and `os`
- `FrozenImporter` - For frozen modules
- `PathFinder` - Modules that are located somewhere in the filesystem like `datetime` or `flask`

Import Flow


import datetime
Iterate over sys.meta_path until a ModuleSpec is returned
ModuleSpec created
Create the Module from spec
No spec found
Module created
Error
Call exec_module
Error
Success
Raise an import error
Return the datetime Module

PathFinder

sys.path

Locations where the `PathFinder` searches for packages/modules

Inside a virtual environment

```
import sys
sys.path
```

sys.path

Locations where the `PathFinder` searches for packages/modules

Inside a virtual environment

```python
import sys
sys.path
```

```
['',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python38.zip',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/lib-dynload',
 '/Users/loz/projects/hacking_import/env/lib/python3.8/site-packages']
```

PYTHONPATH

An environment variable, when the python process starts the contents are appended into `sys.path`

```
export PYTHONPATH=/path/to/foo:/path/to/bar
```

```
sys.path
```

```
['',
 '/path/to/foo',
 '/path/to/bar',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python38.zip',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/lib-dynload',
 '/Users/loz/Library/Python/3.8/lib/python/site-packages',
 '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages']
```

PathFinder

```python
class PathFinder:

    ...

    @classmethod
    def find_spec(cls, fullname, path=None, target=None):
        """Try to find a spec for 'fullname' on sys.path or 'path'.

        The search is based on sys.path_hooks and sys.path_importer_cache.
        """
        if path is None:
            path = sys.path
        spec = cls._get_spec(fullname, path, target)
        if spec is None:
            return None

        # ommitted code
        return spec
```

PathFinder

```python
class PathFinder:

    ...

    @classmethod
    def find_spec(cls, fullname, path=None, target=None):
        """Try to find a spec for 'fullname' on sys.path or 'path'.

        The search is based on sys.path_hooks and sys.path_importer_cache.
        """
        if path is None:
            path = sys.path
        spec = cls._get_spec(fullname, path, target)
        if spec is None:
            return None

        # ommitted code
        return spec
```

PathFinder

```python
class PathFinder:

    ...

    @classmethod
    def find_spec(cls, fullname, path=None, target=None):
        """Try to find a spec for 'fullname' on sys.path or 'path'.

        The search is based on sys.path_hooks and sys.path_importer_cache.
        """
        if path is None:
            path = sys.path
        spec = cls._get_spec(fullname, path, target)
        if spec is None:
            return None

        # ommitted code
        return spec
```

PathFinder

```
class PathFinder:

    ...

    @classmethod
    def find_spec(cls, fullname, path=None, target=None):
        """Try to find a spec for 'fullname' on sys.path or 'path'.

        The search is based on sys.path_hooks and sys.path_importer_cache.
        """
        if path is None:
            path = sys.path
        spec = cls._get_spec(fullname, path, target)
        if spec is None:
            return None

        # ommitted code
        return spec
```

PathFinder

```python
class PathFinder:

    ...

    @classmethod
    def find_spec(cls, fullname, path=None, target=None):
        """Try to find a spec for 'fullname' on sys.path or 'path'.

        The search is based on sys.path_hooks and sys.path_importer_cache.
        """
        if path is None:
            path = sys.path
        spec = cls._get_spec(fullname, path, target)
        if spec is None:
            return None

        # ommitted code
        return spec
```

Extending standard imports

We have already seen everything needed to understand how to extend the import system

Extending standard imports

We have already seen everything needed to understand how to extend the import system

- Create a Finder with the `find_spec` function
- Create a Loader with the `create_module` and `exec_module` functions

Extending standard imports

We have already seen everything needed to understand how to extend the import system

- Create a Finder with the `find_spec` function
- Create a Loader with the `create_module` and `exec_module` functions
- Add the Finder to `sys.meta_path`

Remote Host Finder

Problem: We have a python package that can only be installed on linux hosts. While production does run on linux, in development some of the developers use MacOS.

Solution: Create a new finder - for packages that are not found with the standard finder, it uses a proxy module from a remote linux host.

RPyC Finder

```python
import rpyc
import sys
import importlib

ALLOWED_MODULES = ['linux_only_package',]
class RPyCFinder(importlib.abc.MetaPathFinder):
    def __init__(self):
        self.host = rpyc.classic.connect('ubuntu-host')

    def find_spec(self, fullname, path):
        if path:
            return None
        if fullname not in ALLOWED_MODULES:
            return None

        spec = importlib.util.spec_from_loader(fullname, RPyCLoader(self.host))
        return spec


sys.meta_path.append(RPyCFinder())
```

RPyC Finder

```python
import rpyc
import sys
import importlib

ALLOWED_MODULES = ['linux_only_package',]
class RPyCFinder(importlib.abc.MetaPathFinder):
    def __init__(self):
        self.host = rpyc.classic.connect('ubuntu-host')

    def find_spec(self, fullname, path):
        if path:
            return None
        if fullname not in ALLOWED_MODULES:
            return None

        spec = importlib.util.spec_from_loader(fullname, RPyCLoader(self.host))
        return spec


sys.meta_path.append(RPyCFinder())
```

RPyC Finder

```python
import rpyc
import sys
import importlib

ALLOWED_MODULES = ['linux_only_package',]
class RPyCFinder(importlib.abc.MetaPathFinder):
    def __init__(self):
        self.host = rpyc.classic.connect('ubuntu-host')

    def find_spec(self, fullname, path):
        if path:
            return None
        if fullname not in ALLOWED_MODULES:
            return None

        spec = importlib.util.spec_from_loader(fullname, RPyCLoader(self.host))
        return spec


sys.meta_path.append(RPyCFinder())
```

RPyC Finder

```python
import rpyc
import sys
import importlib

ALLOWED_MODULES = ['linux_only_package',]
class RPyCFinder(importlib.abc.MetaPathFinder):
    def __init__(self):
        self.host = rpyc.classic.connect('ubuntu-host')

    def find_spec(self, fullname, path):
        if path:
            return None
        if fullname not in ALLOWED_MODULES:
            return None

        spec = importlib.util.spec_from_loader(fullname, RPyCLoader(self.host))
        return spec


sys.meta_path.append(RPyCFinder())
```

RPyC Finder

```python
import rpyc
import sys
import importlib

ALLOWED_MODULES = ['linux_only_package',]
class RPyCFinder(importlib.abc.MetaPathFinder):
    def __init__(self):
        self.host = rpyc.classic.connect('ubuntu-host')

    def find_spec(self, fullname, path):
        if path:
            return None
        if fullname not in ALLOWED_MODULES:
            return None

        spec = importlib.util.spec_from_loader(fullname, RPyCLoader(self.host))
        return spec


sys.meta_path.append(RPyCFinder())
```

RPyC Loader

```
1   class RPyCLoader(importlib.abc.Loader):
2       def __init__(self, host):
3           self.host = host
4
5       def create_module(self, spec):
6           proxy_module = getattr(self.host.modules, spec.name)
7           return proxy_module
8
9       def exec_module(self, module):
10          ...
```

RPyC Loader

```
1   class RPyCLoader(importlib.abc.Loader):
2       def __init__(self, host):
3           self.host = host
4
5       def create_module(self, spec):
6           proxy_module = getattr(self.host.modules, spec.name)
7           return proxy_module
8
9       def exec_module(self, module):
10          ...
```

RPyC Loader

```
1   class RPyCLoader(importlib.abc.Loader):
2       def __init__(self, host):
3           self.host = host
4
5       def create_module(self, spec):
6           proxy_module = getattr(self.host.modules, spec.name)
7           return proxy_module
8
9       def exec_module(self, module):
10          ...
```

Module Found

Don’t try this at home!

Problem: Python doesn’t find the modules I ask it to. Whether it was not installed, a spelling mistake in the name or the module just doesn’t exist.

Module Found

Don’t try this at home!

Problem: Python doesn’t find the modules I ask it to. Whether it was not installed, a spelling mistake in the name or the module just doesn’t exist.

Solution: AI

Module Found

Don’t try this at home!

Problem: Python doesn’t find the modules I ask it to. Whether it was not installed, a spelling mistake in the name or the module just doesn’t exist.

Solution: AI

Use Open AI API to generate module functions as they are being called

AI Finder

No, really... I'm not joking!

```python
DISALLOWED_MODULES = ["apport_python_hook", "sitecustomize", "usercustomize"]
class AiFinder():  # pylint: disable=too-few-public-methods
    def __init__(self, api_key: str) -> None:
        self._loader = AiLoader(api_key)

    def find_spec(self, fullname, path, _target=None):
        if path:
            return None
        if '.' in fullname or fullname in DISALLOWED_MODULES:
            return None
        return importlib.util.spec_from_loader(fullname, self._loader)
```

AI Finder

No, really... I'm not joking!

```python
DISALLOWED_MODULES = ["apport_python_hook", "sitecustomize", "usercustomize"]
class AiFinder():  # pylint: disable=too-few-public-methods
    def __init__(self, api_key: str) -> None:
        self._loader = AiLoader(api_key)

    def find_spec(self, fullname, path, _target=None):
        if path:
            return None
        if '.' in fullname or fullname in DISALLOWED_MODULES:
            return None
        return importlib.util.spec_from_loader(fullname, self._loader)
```

AI Finder

No, really... I'm not joking!

```python
DISALLOWED_MODULES = ["apport_python_hook", "sitecustomize", "usercustomize"]
class AiFinder():  # pylint: disable=too-few-public-methods
    def __init__(self, api_key: str) -> None:
        self._loader = AiLoader(api_key)

    def find_spec(self, fullname, path, _target=None):
        if path:
            return None
        if '.' in fullname or fullname in DISALLOWED_MODULES:
            return None
        return importlib.util.spec_from_loader(fullname, self._loader)
```

AI Loader

For your safety, don't do this...

```python
from module_found.lazy_object import LazyModule

class AiLoader():
    def __init__(self, api_key: str) -> None:
        self._api_key = api_key

    def create_module(self, spec):
        return LazyModule(self._api_key, spec.name)

    def exec_module(self, _module):
        ...
```

AI Loader

For your safety, don't do this...

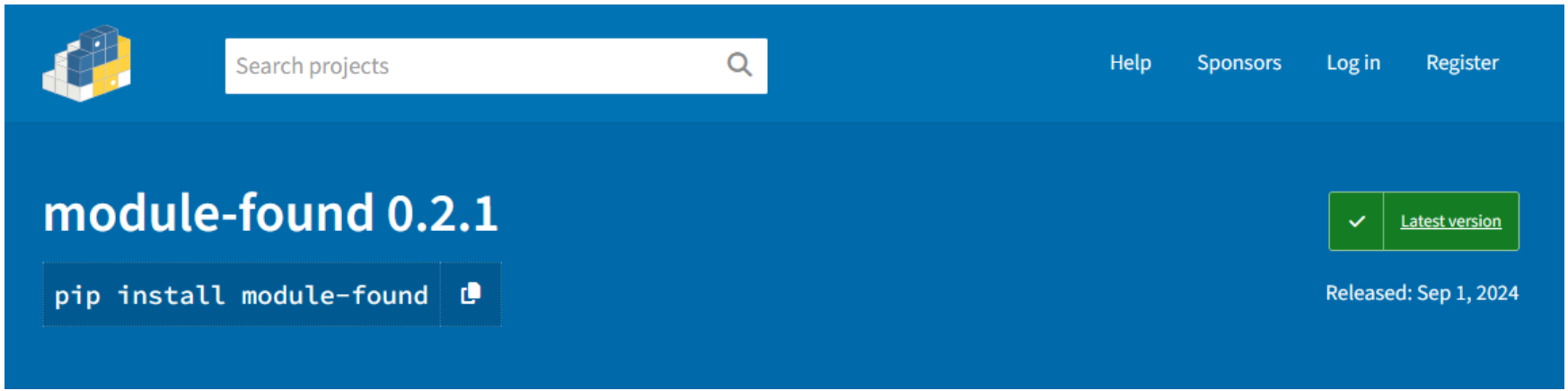
```python
from module_found.lazy_object import LazyModule

class AiLoader():
    def __init__(self, api_key: str) -> None:
        self._api_key = api_key

    def create_module(self, spec):
        return LazyModule(self._api_key, spec.name)

    def exec_module(self, _module):
        ...
```

Module Found Demo

Seriously, DON’T try this at home… Like, ever!

Module Found on PyPI

Alright, here's how you do it at home...



https://pypi.org/project/module-found/

https://github.com/LiadOz/module-found/

But wait, there's more!

- Relative imports - PEP 328
- `sys.path_hooks` - to hook into the PathFinder code
- Lazy imports in Python 3.12 PEP-690