How import works in Python

戴嘉华 *created on* MAY 12, 14 *last updated on* MAY 21, 14

When come to python's `import`, there are some confusions about it causing headache.

Key Takeaways:

- Python uses `__name__` to resolve relative path of import statement
- Python uses `sys.path` to resolve the path of absolute import statement

---

Python uses `__name__` to resolve relative path of import statement

Suggest that we have application structure like this:

```
phone/
    __init__.py
    app.py
    sound/
        __init__.py
        echo.py
    display/
        __init__.py
        show.py
```

All `__init__.py`s are simply set to be empty. Codes in `app.py`, `echo.py`, `show.py` are listed below:

app.py

```
from sound import echo
```

echo.py

```
from ..display import show
```

show.py

```
print 'Showing Screen !!'
```

The most common mistake newbie usually makes is run `python app.py` command on the top level of the package, which will cause an error and ruthlessly throw you onto the wall:

```
Traceback (most recent call last):
  File "app.py", line 1, in <module>
    from sound import echo
  File "/home/livoras/phone/sound/echo.py", line 1, in <module>
    from ..display import show
ValueError: Attempted relative import beyond toplevel package
```

That simple because the import mechanism of python is that it uses `__name__` to resolve module's path while you import it in relative way. For example, if the `__name__` of a module is `package1.package2.module3`, and module3 imports module in upper package using `from ..packge3 import module4`, then python will combine `package1.package2.module3` with `..package3` to get `package1.package3` which locates the package `package3` and then looks for `module4` module inside of it.

So, what's the reason for exception above then? Let's check the `__name__` of each module we've mentioned. Execute `app.py` in the same way.

app.py

```
print __name__ # __main__
from sound import echo
```

echo.py

```
print __name__ # sound.echo
from ..display import show
```

show.py

```
print __name__ # Never execute to here for exception sake.
print 'Showing Screen !!'
```

Take a look at `__name__` of `echo.py`. Python is looking for `display/show`, which is the result of combination of `sound.echo` and `..display`. But that path is not avaible because it supposes to be `phone/display/show`.

And how to solve it?

Simply add a python script outside the top package and let it import `app.py`:

outside.py

```
from phone import app
```

Why? Look at the `__name__` of each file. The `__name__` of `app.py` is `phone.app` and for `echo.py` is `phone.sound.echo`. So, result of `from ..display import show` would be `phone.display.show`, and this path does exist!

So, that leads to a conclusion:

If you want to use relative import statement with dot notation (`..`), you have to have your sub packages' parent package imported by other module instead of being executed directly.

References:

- http://stackoverflow.com/questions/72852/how-to-do-relative-imports-in-python
- http://legacy.python.org/dev/peps/pep-0328/
- https://docs.python.org/2/tutorial/modules.html#packages

(The End)

Tags: python