Python计算生态构建

嵩 天

Python应用基础 版权 by 嵩天

模块和包的构建

嵩 天

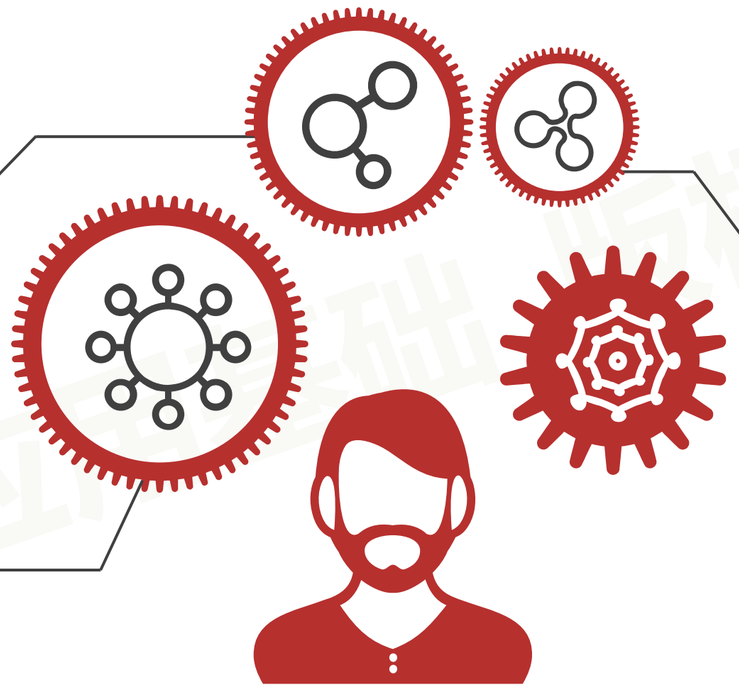
模块和包的构建

单元开篇

单元开篇

(2) 常规包的构建

(1) 模块的构建



(3) 命名空间包的构建

Python应用基础 版权 by 嵩天

模块和包的构建

模块和包的构建

模块的构建

模块的构建

模块的构建原则：如何编写好一个.py文件?

- 功能闭包：单一.py文件实现单一且完整的功能
- 抽象适度：用函数或类进行抽象，结合功能选择合适抽象
- 操作闭包：模块无顶层可执行语句，导入时无输出

Python应用基础 版权 by 嵩天

模块的构建：不建议的方式

```
module_var = 1
class module_class:
    mc_classattr = 1
    def __init__(self, mc_instattr = 1):
        self.mc_instattr = mc_instattr
    def mc_func(self):
        return "Method with a count of {}".format(self.mc_classattr)

def module_func():
    print("Module Function")

print("Module Statement")
```

类对象
无输出

模块的构建：建议的方式

```
class module_class:
    mc_classattr = 1
    def __init__(self, mc_instattr = 1):
        self.mc_instattr = mc_instattr
    def mc_func(self):
        return "Method with a count of {}".format(self.mc_classattr)
    def module_func():
        print("Module Function")

if __name__ == "__main__":
    module_var = 1
    print("Module Statement")
```

模块的构建

模块的构建原则

- 功能闭包：功能定义要清晰、设计要合理（紧耦合vs松耦合）
- 抽象适度：采用类或函数，尽量选择一种；多种也无妨
- 操作闭包：采用__name__、无全局可执行语句、尽量无全局变量

Python应用基础 版权 by 嵩天

模块的构建：扩展实例

```
"这是模块描述"
class module_class:
    mc_classattr = 1
    def __init__(self, mc_instattr = 1):
        self.mc_instattr = mc_instattr
    def mc_func(self):
        return "Method with a count of {}".format(self.mc_classattr)

if __name__ == "__main__":
    import sys
    mc = module_class(sys.argv[1])
    print("Module Statement")
```

第一行增加描述

顶层功能的引用放在__name__后

模块的构建

dir()函数：以列表形式返回模块所使用的命名

```
import m
print(dir(m))
```

```
['__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__',
'module_class']
```

模块和包的构建

常规包的构建

包的构建

包有两种：常规包 和 命名空间包

- 常规包：Regular Packages，通过__init__.py对文件和目录组织形成的包
- 命名空间包：Namespace Packages，由更分散子包组成的包
  - 子包的位置可以在文件系统中不连续
  - 子包可以是压缩文件或网络连接或其他系统资源

Python应用基础 版权 by 嵩天

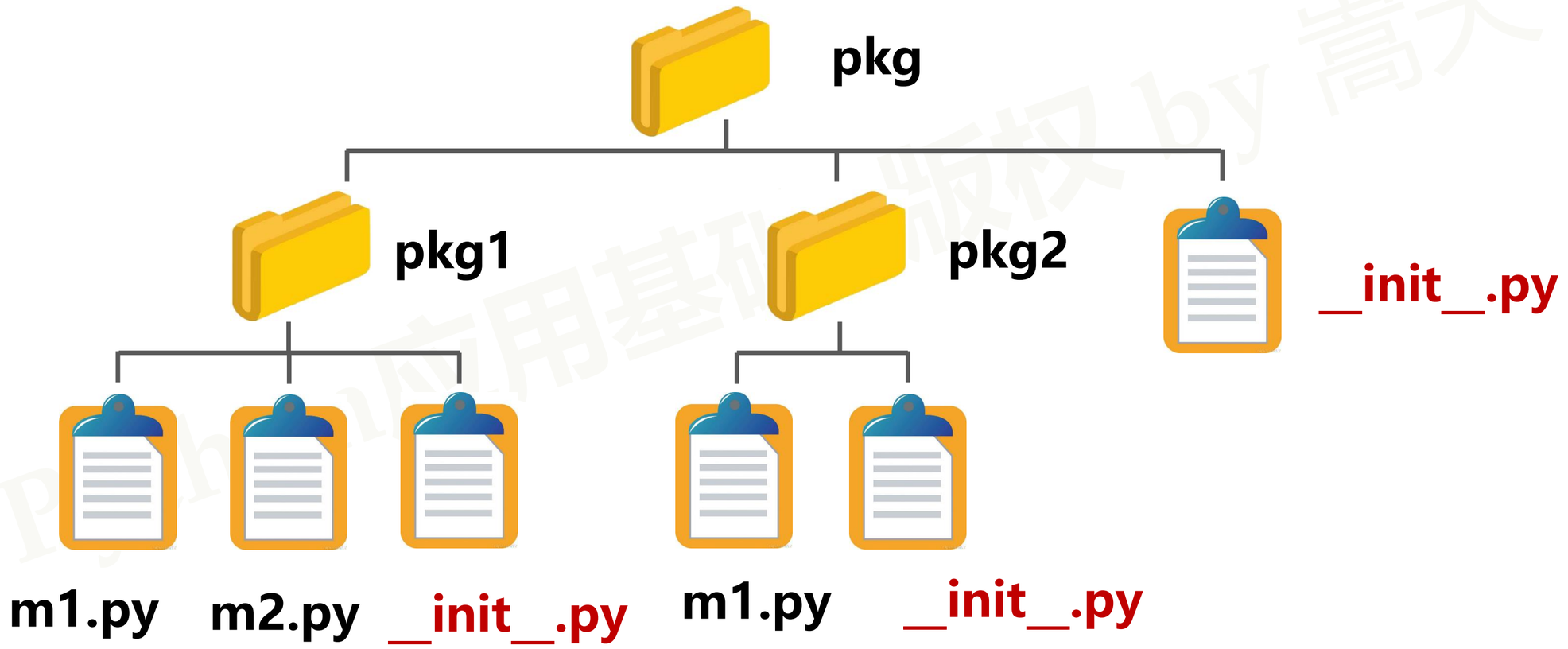
常规包的构建

常规包 Regular Packages

- 连续目录空间表达的、有层次的命名空间
- 每个目录中包含一个__init__.py文件，可以是空文件
- 当包/子包被导入时，对应目录的__init__.py文件将被执行

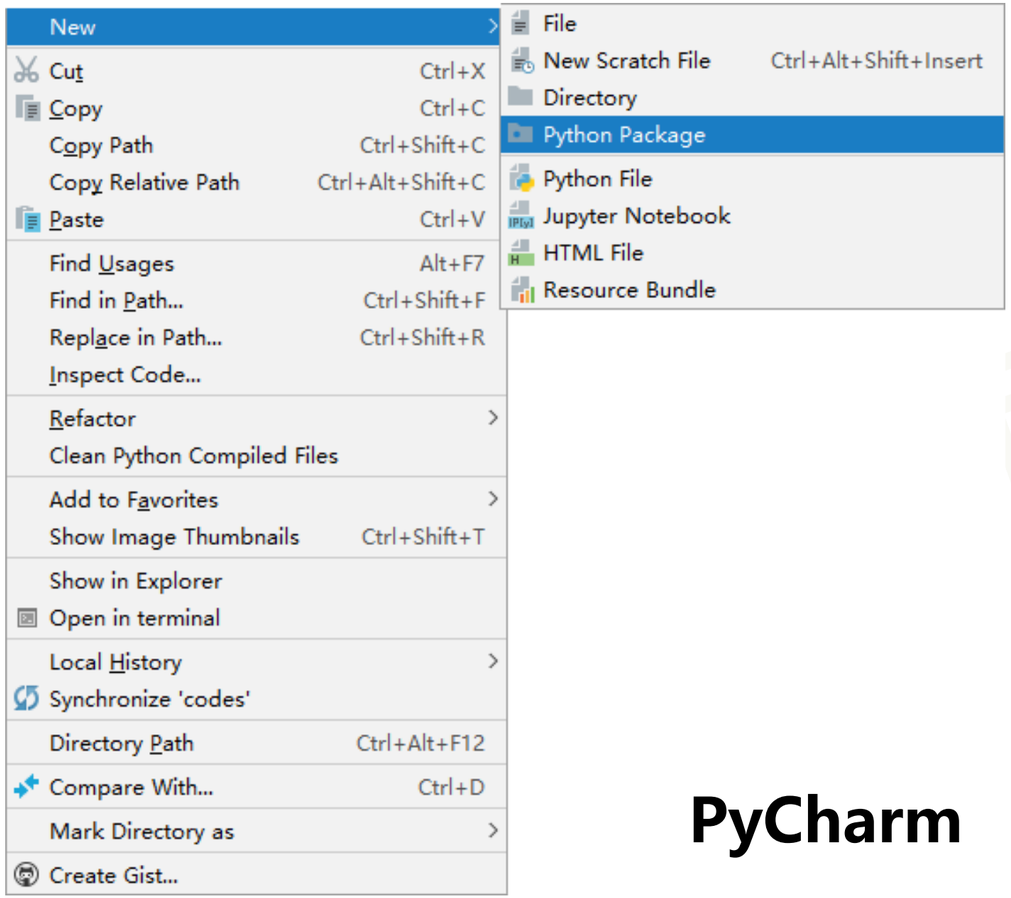
Python应用基础 版权 by 嵩天

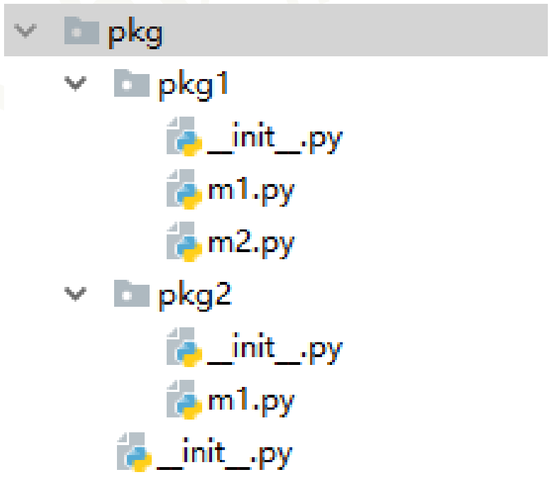
常规包构建实例



Python应用基础 版权 by 嵩天

常规包构建实例





PyCharm

常规包构建实例

pkg1/m1.py

```
'pkg1 -> m1'
def mecho(any):
    print("pkg1-m1: {}".format(any))
```

pkg1/m2.py

```
'pkg1 -> m2'
def mecho(any):
    print("pkg1-m2: {}".format(any))
```

pkg2/m1.py

```
'pkg2 -> m1'
def mecho(any):
    print("pkg2-m1: {}".format(any))
```

pkg/__init__.py

```
print("[INFO]: 导入pkg模块")
```

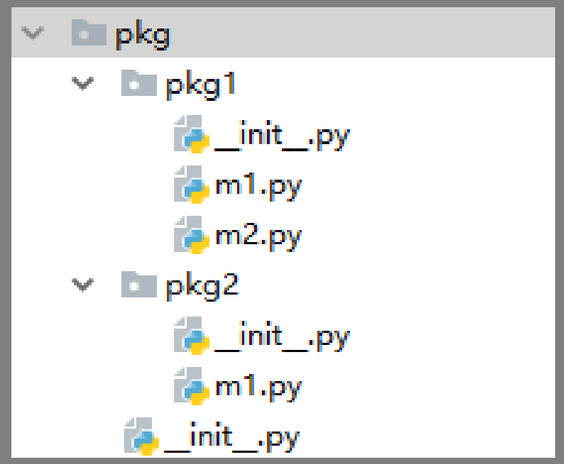
pkg/pkg1/__init__.py

```
print("[INFO]: 导入pkg1模块")
```

pkg/pkg2/__init__.py

```
print("[INFO]: 导入pkg2模块")
```

常规包构建实例分析（1）



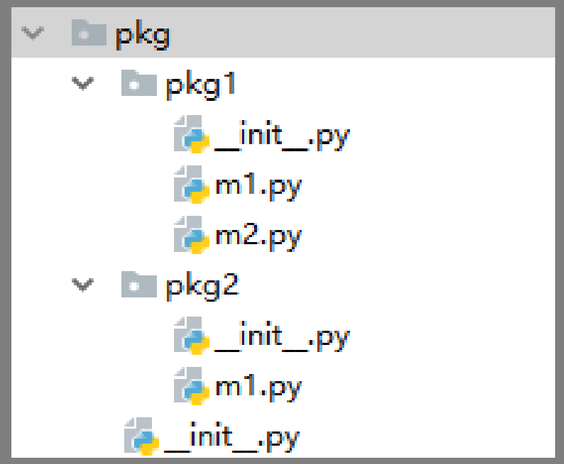

```
import pkg.pkg1.m1
import pkg.pkg1.m2
import pkg.pkg2.m1

pkg.pkg1.m1.mecho(123)
pkg.pkg1.m2.mecho("Python123")
pkg.pkg2.m1.mecho(["Python",123])
```

```
[INFO]: 导入pkg模块
[INFO]: 导入pkg1模块
[INFO]: 导入pkg2模块
pkg1-m1: 123
pkg1-m2: Python123
pkg2-m1: ['Python', 123]
```

每个包仅被导入一次，且包导入按照层次结构进行

常规包构建实例分析（2）



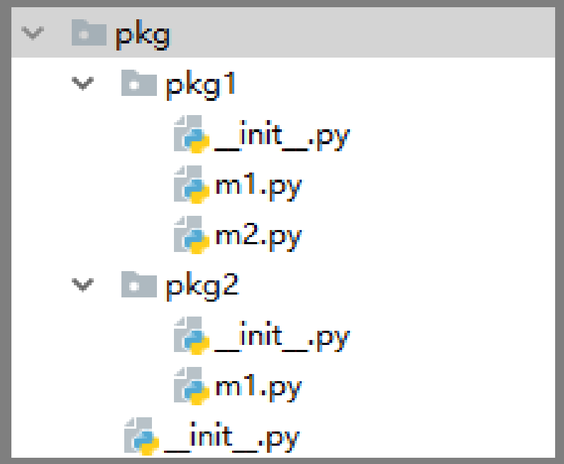

```python
import pkg.pkg1

pkg.pkg1.m1.mecho(123)
pkg.pkg1.m2.mecho("Python123")
```

```
Traceback (most recent call last):
  File "...", line 5, in <module>
    pkg.pkg1.m1.mecho(123)
AttributeError: module 'pkg.pkg1' has no attribute 'm1'
```

直接导入包不行，需要进行到模块层次

常规包构建实例分析（3）




```
from pkg.pkg1 import m1,m2

m1.mecho(123)
m2.mecho("Python123")
```

[INFO]：导入pkg模块

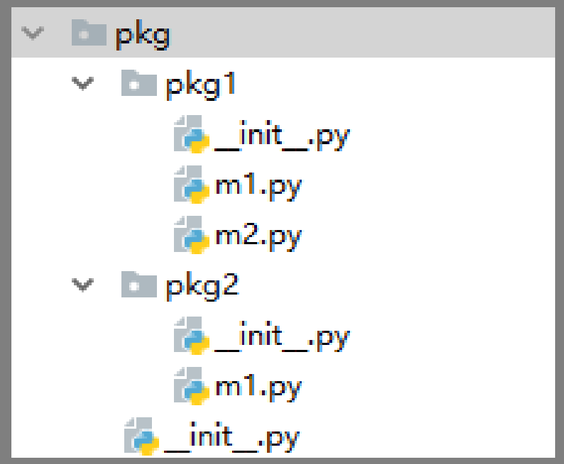
[INFO]：导入pkg1模块

pkg1-m1：123

pkg1-m2：Python123

from..import直接导入具体模块，可以简化调用时命名空间表达

常规包构建实例分析（4）




```
from pkg.pkg1 import *

m1.mecho(123)
m2.mecho("Python123")
```

```
Traceback (most recent call last):
  File "...", line 5, in <module>
    m1.mecho(123)
NameError: name 'm1' is not defined
```

from..import * 需要额外代码来编写

常规包的构建

__all__ 属性

- from…import * 形式需要在__init__.py文件中增加__all__属性赋值
- __all__需要被赋值为列表对象，包含当前包下所有希望被导入的模块名称
- __all__用来辅助导入模块，但不能辅助导入列表

Python应用基础 版权 by 嵩天

常规包构建实例

pkg1/m1.py

```
'pkg1 -> m1'
def mecho(any):
    print("pkg1-m1: {}".format(any))
```

pkg1/m2.py

```
'pkg1 -> m2'
def mecho(any):
    print("pkg1-m2: {}".format(any))
```

pkg2/m1.py

```
'pkg2 -> m1'
def mecho(any):
    print("pkg2-m1: {}".format(any))
```

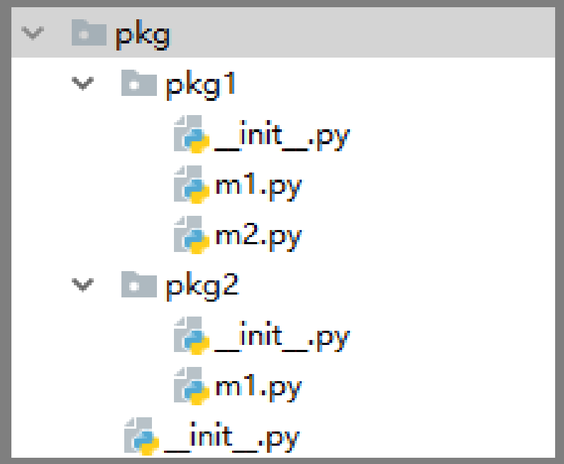
pkg/pkg1/__init__.py

```
print("[INFO]: 导入pkg1模块")
__all__ = ['m1', 'm2']
```

pkg/pkg2/__init__.py

```
print("[INFO]: 导入pkg2模块")
__all__ = ['m1']
```

常规包构建实例分析（5）




```
from pkg.pkg1 import *

m1.mecho(123)
m2.mecho("Python123")
```

[INFO]: 导入pkg模块

[INFO]: 导入pkg1模块

pkg1-m1: 123

pkg1-m2: Python123

__all__属性支持from..import * 功能

常规包的构建

常规包的构建流程

- 设计好包层次结构（命名空间），每个目录配置一个__init__.py文件
- 完善__init__.py文件，并酌情赋值__all__属性
- 再次理解包和模块的不同，模块提供功能，包提供命名空间

Python应用基础 版权 by 嵩天

常规包的构建

常规包的导入流程（import）

- 每个包仅被导入一次，且包导入按照层次结构进行
- 直接导入包不能调用功能，需要导入到模块层次
- from..import直接导入具体模块，可以简化调用时命名空间表达
- __init__.py文件中通过__all__属性支持from..import * 功能

Python应用基础 版权 by 嵩天

模块和包的构建

命名空间包的构建

包的构建

包有两种：常规包 和 命名空间包

- 常规包：Regular Packages，通过__init__.py对文件和目录组织形成的包
- 命名空间包：Namespace Packages，由更分散子包组成的包
  - 子包的位置可以在文件系统中不连续
  - 子包可以是压缩文件或网络连接或其他系统资源

Python应用基础 版权 by 嵩天

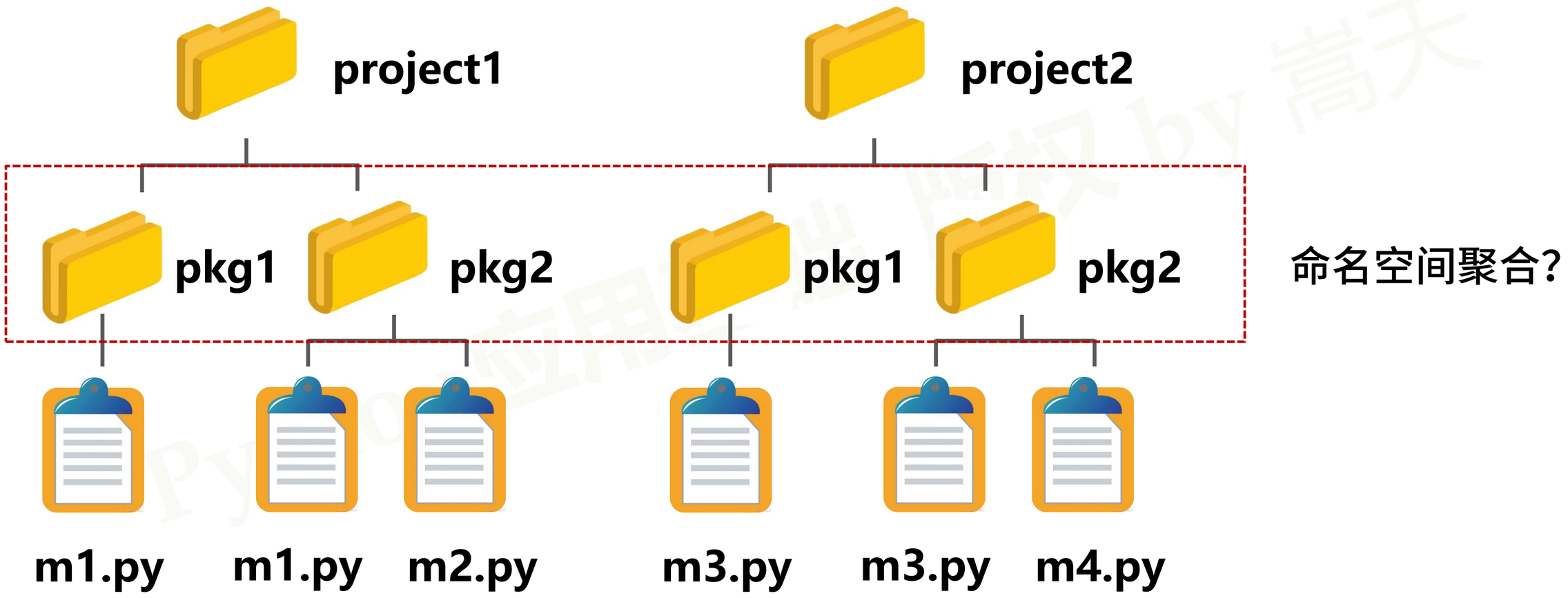
命名空间包

命名空间包：表达命名空间层次结构的一种逻辑包形式

- 命名空间中各部分可以在不同的文件系统位置
- 命名空间中各子包并不包含__init__.py文件 （普通目录！）
- Python解释器通过sys.path变量来隐式维护命名空间包

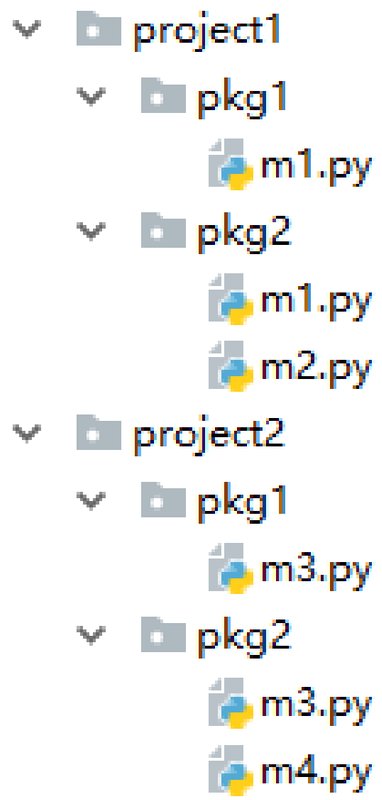
Python应用基础 版权 by 崔天

命名空间包构建实例



Python应用基础 版权 by 嵩天

命名空间包构建实例



- 全部是正常目录，无__init__.py文件
- 最外层目录不同，其中子包名称相同

应用基础 版权 by 嵩天

命名空间包构建实例

project1/pkg1/m1.py

```
def mecho(any):
    print("pkg1-m1: {}".format(any))
```

project2/pkg1/m3.py

```
def mecho(any):
    print("pkg1-m3: {}".format(any))
```

project1/pkg2/m1.py

```
def mecho(any):
    print("pkg2-m1: {}".format(any))
```

project2/pkg2/m3.py

```
def mecho(any):
    print("pkg2-m3: {}".format(any))
```

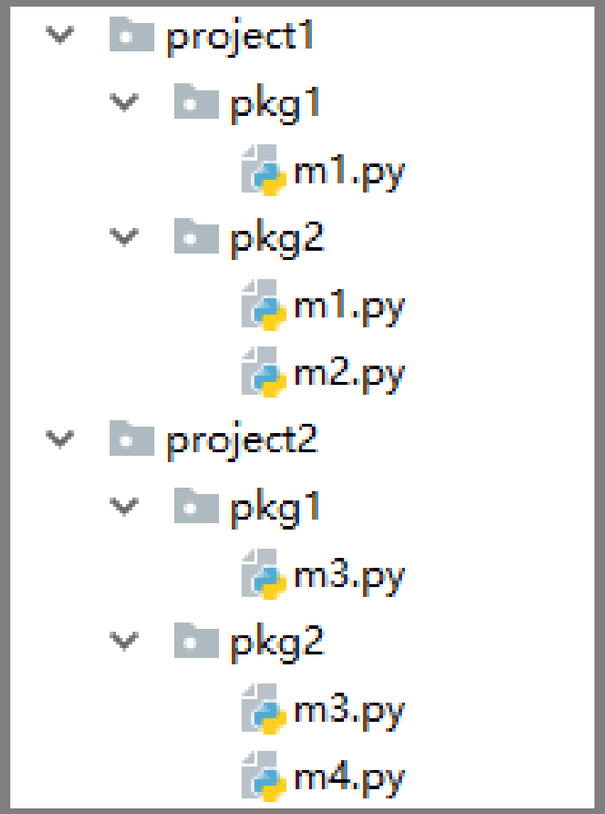
project1/pkg2/m2.py

```
'pkg2 -> m1'
def mecho(any):
    print("pkg2-m2: {}".format(any))
```

project2/pkg2/m4.py

```
'pkg2 -> m1'
def mecho(any):
    print("pkg2-m4: {}".format(any))
```

命名空间包构建实例分析



test.py

```python
import sys

sys.path += ['project1', 'project2']

import pkg1.m1
import pkg1.m3

pkg1.m1.mecho(123)
pkg1.m3.mecho("Python")

print(pkg1.__path__)
```

命名空间包构建实例分析

```
import sys

sys.path += ['project1', 'project2']

import pkg1.m1
import pkg1.m3

pkg1.m1.mecho(123)
pkg1.m3.mecho("Python")

print(pkg1.__path__)
```

将最顶层目录加入sys.path变量

命名空间包构建实例分析

```
import sys

sys.path += ['project1', 'project2']

import pkg1.m1
import pkg1.m3

pkg1.m1.mecho(123)
pkg1.m3.mecho("Python")

print(pkg1.__path__)
```

顶层目录内结构成为一个命名空间

```
pkg1-m1: 123
pkg1-m3: Python
```

命名空间包构建实例分析

```python
import sys

sys.path += ['project1', 'project2']

import pkg1.m1
import pkg1.m3

pkg1.m1.mecho(123)
pkg1.m3.mecho("Python")

print(pkg1.__path__)
```

获得pkg1命名空间内变量

```
_NamespacePath(['project1\\pkg1',
                'project2\\pkg1'])
```

命名空间包

sys.path：指定搜索路径的字符串列表

- 指定import时搜索模块或包的路径列表，路径是相对或绝对路径
- sys.path是一个列表类型，可以用sys.path.append(p)增加新路径p
- 载入后，根据其中包的名称和层次结构自然组成了命名空间包

Python应用基础 版权 by 嵩天

命名空间包

__path__属性

- 记录了某个包（命名空间）的绝对或相对路径，列表类型
- 常规包：路径是单一的，列表中只有一个元素
- 命名空间包：路径是多元的，列表中可能有多个元素

pkg1命名空间的路径：['project1\\pkg1', 'project2\\pkg1']

Python应用基础 版权 by 嵩天

模块和包的构建

单元小结

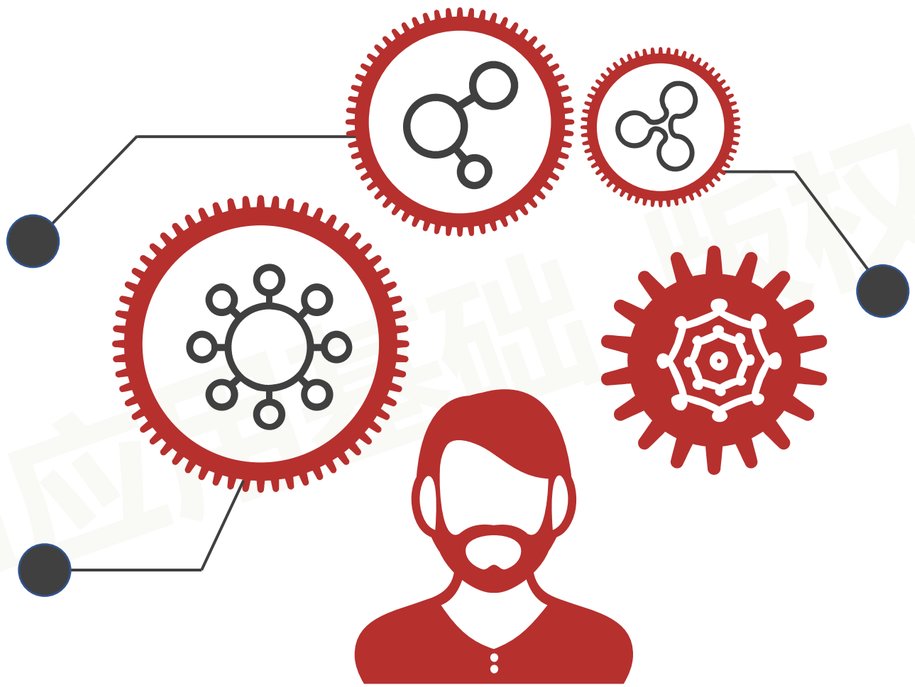
单元小结

(2) 常规包的构建

__init__.py、__all__属性

(1) 模块的基本构建

构建原则、__name__、dir()



(3) 命名空间包的构建

sys.path、__path__

模块和包的构建

Python应用基础 版权 by 嵩天

Python 123

Thank you