Python程序设计

第六讲 函数与模块化

模块和包

张 华
WHU

模块和包

- 模块
- 模块化编程
- 包
- 标准库

再看程序的模块化

- **用模块来组织程序的代码**
  - 如果程序中包含多个可以复用的函数或类，则通常把相关的函数和类分组包含在单独的模块（module）中。
  - 这些提供计算功能的模块称之为模块（或函数模块），导入并使用这些模块的程序，则称之为客户端程序。
- **模块化编程**
  - 把计算任务分离成不同模块的程序设计方法，称之为模块化编程（modular programming）。
  - 使用模块，可以将计算任务分解为大小合理的子任务，并实现代码的重用。

模块化编程

- **模块API**
  - 客户端使用模块提供的函数时，无须了解其实现细节。
  - 模块和客户端之间遵循的契约称之为**API**（**Application Programming Interface**，应用程序编程接口）。
  - **API**用于描述模块中提供的函数的功能和调用方法。
- **模块化程序设计的基本原则**
  - 先设计**API**（即模块提供的函数或类的功能描述），
  - 然后实现**API**（即编写程序，实现模块函数或类），
  - 最后在客户端中导入并使用这些函数或类。

模块化编程

模块API

- 除了Python文档，通过内置函数help()，也可以查看Python模块的API。

  import <模块名>

  help(模块名)

- 举例

```
>>> import math
>>> help(math)

Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
…
```

模块化编程

模块的实现

- “实现”是指编写函数或类的代码，模块的实现就是若干函数或类的代码集合，保存在一个后缀为.py的文件中。
- 模块的实现必须遵循API规约，可以采用不同算法实现API，这为模块的改进和版本升级提供了无缝对接。新实现遵循API，则客户端程序无须修改也可以正常运行。

模块化编程

模块的客户端

- 客户端遵循API提供的调用接口，导入和调用模块中实现的函数功能。
- API允许任何客户端直接使用模块，而无需检测模块中定义的代码。
  - 例如，可以直接使用模块math和random。

模块化编程

模块设计的一般原则

- （1）先设计API，再实现模块。
- （2）控制模块的规模，只为客户端提供需要的函数。
  - 实现包含大量函数的模块会导致模块的复杂性。
  - 例如，Python的math模块中就不包含正割函数、余割函数和余切函数，因为这些函数很容易通过函数math.sin()、math.cos()和math.tan()的计算而得。
- （3）在模块中编写测试代码，并消除全局代码。
- （4）使用私有函数实现不被外部客户端调用的模块函数。
- （5）通过文档提供模块帮助信息。

模块化编程

设计模块API

- API定义客户端和实现之间的契约。
- API是一个明确的规范，规定“实现”的具体功能是什么。
- API通常由两部分组成：
  - 可用函数的签名的精确规范，
  - 以及描述函数作用的非正式自然语言描述。
- API一般使用表格的形式，描述模块中的变量、函数和类。
  - 举例：设计包含4种算术运算的模块mymath

函数调用	功能描述
add(x, y)	加法函数 add(x, y)
sub(x, y)	减法函数 sub(x, y)
mul(x, y)	乘法函数 mul(x, y)
div(x, y)	除法函数 div(x, y)

模块化编程

创建模块

- Python模块对应于包含Python代码的源文件（其扩展名为.py），在文件中可以定义变量、函数和类。
- 在模块中，除了可以定义变量、函数和类之外，还可以包含一般的语句，称之为主块（全局语句）。当运行该模块，或导入该模块时，主块语句将依次执行。
- 例如，mymath.py

```
PI = 3.14

def add(x, y):
    return x + y

def sub(x, y):
    return x - y

def mul(x, y):
    return x * y

def div(x, y):
    return x / y
```

模块化编程

Python中的内置模块

- 模块名：__builtins__
- 该模块不需手动导入，启动Python时系统会自动导入，任何程序都可以直接使用它们。
- 该模块定义了一些软件开发中常用的函数，实现了数据类型转换， 数据计算， 序列的处理、常用字符串处理等。
- 内置模块中的函数称内置函数（有时又称系统函数）。
  - 内置函数使用时不需加模块名前缀。
  - 常用的内置函数举例：
    - print()
    - input()
    - type()
    - dir()

模块化编程

Python中的内置模块

用dir()函数查看内置模块的成员

- dir(__builtins__)

```
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferEr
ror', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'Conne
ctionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsErr
or', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarni
ng', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'Lookup
Error', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplement
edError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionEr
ror', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'Syn
taxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalE
rror', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarni
ng', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '_', '__build_class__', '__debug__', '__doc__', '__im
port__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray',
'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divm
od', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'ha
sh', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map',
'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr'
, 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'var
s', 'zip']
```

模块化编程

导入非内置模块并使用

- Python中使用如下语句来导入模块：

  import 模块名

  - 其中模块名也可以有多个，多个模块之间用逗号分隔。该语句通常放在程序的开始部分。
- 可使用内置函数dir()函数检查当前已导入的模块。
- 使用dir(模块名)可查看模块的内部，如: dir(math)。
- 模块导入后，在程序中使用“模块名.函数名”形式来使用其中的函数。
  - 例如：
    - 首先导入math模块：import math。
    - 然后执行以下语句计算(a2+b2)的平方根：
    - print(math.sqrt(a*a+b*b))。

模块化编程

导入非内置模块并使用

- 为了简化，可以使用以下语句代替import语句：

  from 模块名 import *

  - 这样，在调用函数时可以省略“模块名.”前缀。
  - 例如：
    - **from math import ***
    - **sqrt(3)**
  - 但要注意所引入模块中的函数名等不能与现有系统中的产生冲突。

模块化编程

导入非内置模块的三种方法

导入模块
    import <模块名>
调用函数（或常量）
    <模块名>.<函数或常量>

导入模块的函数
    from <模块名> import <函数名>
调用该函数
    <函数>

导入模块中的所有函数
    from <模块名> import *
调用函数（或常量）
    <函数或常量>

举例

```
>>>from math import sqrt   #引入数学库中的sqrt函数
>>>sqrt(16)
4.0
>>>from math import *      #引入数学库中所有的函数
>>>sqrt(16)
4.0
```

模块化编程

模块的__name__属性

- 每个.py文件在运行时都有一个__name__属性。
  - 如果作为模块导入，该属性值为模块名；
  - 如果文件独立运行，该属性值为“__main__”。
- 通过对其属性值的判断，使文件在两种情况下都能正确使用。
- 举例：在mymath.py种添加以下代码

```
def test_mymath():
    print(add(12, 34))
    print(sub(90, 80))
    print(mul(6, 7))
    print(div(9, 8))

if __name__ == '__main__':
    test_mymath()
```

模块化编程

模块的__name__属性的应用举例

- 使用方式1：直接运行mymath.py文件，对模块中的函数进行测试。

```
(base) D:\Course\pycourse>python mymath.py
46
10
42
1.125
```

- 使用方式2：导入mymath模块，使用其中的函数。

```
>>> import mymath
>>> print(mymath.add(123, 456))
579
```

模块化编程

模块的私有函数

- 实现模块时，有时候需要在模块中定义仅在模块中使用的辅助函数。辅助函数不提供给客户端直接调用，故称之为私有函数。
- 按惯例，Python程序员使用下划线开始的函数名作为私有函数的名称。私有函数客户端不应该直接调用，故API中不包括私有函数。
- Python语言没有强制不允许调用私有函数的机制，程序员应该避免直接调用私有函数。

模块化编程

编写模块的文档字符串

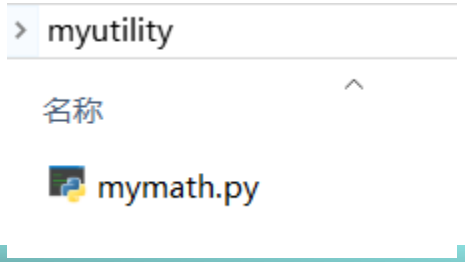
- 在函数的第一个逻辑行的字符串称为函数的文档字符串，用于提供有关函数的帮助信息。
- 文档字符串一般遵循下列惯例：
  - 文档字符串是一个多行字符串；
  - 首行以大写字母开始，句号结尾；
  - 第二行是空行；
  - 从第三行开始是详细的描述。
- 可以使用三种方法抽取函数的文档字符串帮助信息：
  - 使用内置函数：**help(函数名)**；
  - 使用函数的特殊属性：函数名.__doc__；
  - 第三方自动化工具也可以抽取文档字符串信息，以形成帮助文档。

模块化编程

- .pyc文件与Python程序执行
  - .pyc文件是经过编译后的字节码，这样下次导入时，如果模块源代码.py文件没有修改（通过比较两者的时间戳），则直接导入.pyc文件，从而提高程序效率。
  - 按字节编译的.pyc文件是在导入模块时，python解释器自动完成，无需程序员手动编译。

进一步用“包”组织程序代码

- 包
  - 功能相似的模块使用包组成层次组织结构。
- 创建包
  - 在指定目录中创建对应包名的目录。
  - 在该目录下创建一个特殊文件：__init__.py。（非必须）
    - 可以记录该包中的模块，内容可以为空。
  - 最后在该目录下创建模块文件。
  - 例如，把模块mymath放到包myutility中
    - 创建文件夹myutility，把mymath.py移动到该文件夹中



包

导入和使用包中的模块

- 导入包中的成员的方式

  import <包名1[.包名2...]>.<模块名>

  from <包名1[.包名2...]> import <模块名>

  from <包名1[.包名2...]> import <模块名>.<成员名>

  from <包名1[.包名2...]>.<模块名> import <成员名>

  from <包名1[.包名2...]> import *

- 举例，使用myutility包中mymath模块的add函数

```
>>> from myutility import mymath
>>> print(mymath.add(12, 34))
46
```

模块的导入顺序

- 导入模块时，解释器按下列目录搜索路径和文件搜索顺序查找并导入文件。
  - （1）当前目录。
    - 启动交互式Python的目录，或Python主程序位于的目录。
  - （2）操作系统环境变量中指定的目录。
  - （3）Python标准库目录。

命名空间与名称查找顺序

- 当代码中使用名称x时，Python解释器把x解释为对象名（对象、函数、变量等），并按如下命名空间顺序查找以x命名的对象：
  - （1）局部命名空间。当前函数或类的方法中定义的局部变量。
  - （2）全局命名空间。当前的模块（.py文件）中定义的变量、函数或类。
  - （3）内置命名空间。对每个模块都是全局的。作为最后的尝试，Python将假设x是内置函数或变量。

Python标准库

Python标准库

- 随着每个Python版本的发布，会同时发布该版本的Python标准库。
- Python的标准库中包含很多模块，为操作系统、解释器和互联网之间的交互提供了丰富的工具。其中既有Python语言自身特定的类型和声明，也包含一些只用于少数程序的模块。所有这些模块都得到充分测试，可以用来作为应用开发的起点。

Python标准库

Python标准库中的基础模块

- 支持内建数据类型操作的模块，如前面提到的用于数学计算操作的math模块；为复数提供类似操作的cmath模块；以及实现常用字符串处理的string模块等。
- os模块：包含了常用的操作系统功能。
- sys模块：通过该模块可以访问程序解释器相关参数，如解释器版本号、模块搜索路径等。
  - 例如：print(sys.path)
- datetime和time模块：用于处理日期时间。
- random模块：提供了产生随机数（以及随机字符）的多种方法。