

Python3 标准库

Stephen CUI

2023-09-20

Contents

1	算法	3
1.1	functools: 管理函数的工具	3
1.1.1	缓存	3
1.1.2	归约数据集	3
1.1.3	泛型函数	3
1.2	itertools: 迭代器函数	4
1.2.1	合并和分解迭代器	4
1.2.2	转换输入	4
1.2.3	生成新值	4
1.2.4	过滤	5
1.2.5	数据分组	5
1.2.6	合并输入	5
1.3	operator: 内置操作符的函数接口	5
1.3.1	逻辑操作	6
1.3.2	比较操作符	6
1.3.3	算术操作符	6
1.3.4	序列操作符	6
1.3.5	就地操作符	6
1.3.6	属性和元素“获取方法”	6
1.3.7	结合操作符和定制类	6

Chapter 1

算法

Python 包含很多模块，可以采用最适合任务的方式来精巧而简洁地实现方式。它支持不同的编程方式，包括纯过程式、面向对象式和函数式。这 3 中方式经常在同一个程序中不同部分混合使用。

`functools` 包含的函数用于与创建函数修饰符、启用面向方面（`aspect-oriented`）编程以及传统面向对象方法不能支持的代码重用。它还提供一个类修饰符以使用一个快捷方式来实现所有富比较 API，另外提供了 `partial` 对象来创建函数（包含其参数）的引用。

`itertools` 模块包含的函数用于创建和处理函数式编程中使用的迭代器和生成器。通过提供基于函数的内置操作接口（如算术操作和元素查找）。

`operator` 模块在使用函数式编程时不在需要很多麻烦的 `lambda` 函数。

不论使用哪一种编程方式，`contextlib` 都会让资源管理更容易、更可靠且更简洁。结合上下文管理器和 `with` 语句，可以减少 `try:finally` 块的个数和所需的缩进层次，同时还能确保文件、套接字、数据库事务和其他资源在适当的时候关闭和释放。

1.1 `functools`: 管理函数的工具

1.1.1 缓存

The `lru_cache()` decorator wraps a function in a “least recently used” cache. Arguments to the function are used to build a hash key, which is then mapped to the result. Subsequent calls with the same arguments will fetch the value from the cache instead of calling the function. The decorator also adds methods to the function to examine the state of the cache (`cache_info()`) and empty the cache (`cache_clear()`).

1.1.2 归约数据集

The `reduce()` function takes a callable and a sequence of data as input. It produces a single value as output based on invoking the callable with the values from the sequence and accumulating the resulting output.

1.1.3 泛型函数

在类似 Python 的动态类型语言中，通常需要基于参数的类型完成稍有不同的操作，特别是在处理元素列表与单个元素的差别时。直接检查参数的类型固然很简单，但是有些情况下，行为差异可

能需要被隔离到单个的函数中，对于这些情况，`functools` 提供了 `singledispatch()` 修饰符来注册一组泛型函数（generic function），可以根据函数第一个参数的类型自动切换。

1.2 itertools: 迭代器函数

`itertools` 包括一组用于处理序列数据集的函数。

1.2.1 合并和分解迭代器

`chain()` 函数取多个迭代器作为参数，最后返回一个迭代器，它会生成所有输入迭代器的内容，就好像这些内容来自一个迭代器一样。利用 `chain()`，可以轻松地处理多个序列而不必构造一个很大的列表。

If the iterables to be combined are not all known in advance, or if they need to be evaluated lazily, `chain.from_iterable()` can be used to construct the chain instead.

The built-in function `zip()` returns an iterator that combines the elements of several iterators into tuples. As with the other functions in this module, the return value is an iterable object that produces values one at a time. `zip()` stops when the first input iterator is exhausted. To process all of the inputs, even if the iterators produce different numbers of values, use `zip_longest()`. 也就是说最短的那个迭代器迭代完就结束，只要遇到 `StopIteration` 就结束。By default, `zip_longest()` substitutes `None` for any missing values. Use the `fillvalue` argument to use a different substitute value.

The `islice()` function returns an iterator that returns selected items from the input iterator, by index. `islice()` takes the same arguments as the slice operator for lists: start, stop, and step. The start and step arguments are optional.

The `tee()` function returns several independent iterators (defaults to 2) based on a single original input. `tee()` 返回的迭代器可以用来为并行处理的多个算法提供相同的数据集。`tee()` 创建的新迭代器会共享器输入迭代器，所有创建了新迭代器后，不应再使用原迭代器。

1.2.2 转换输入

The built-in `map()` function returns an iterator that calls a function on the values in the input iterators, and returns the results. It stops when any input iterator is exhausted.

The `starmap()` function is similar to `map()`, but instead of constructing a tuple from multiple iterators, it splits up the items in a single iterator as arguments to the mapping function using the `*` syntax. Where the mapping function to `map()` is called `f(i1,i2)`, the mapping function passed to `starmap()` is called `f(*i)`.

1.2.3 生成新值

The `count()` function returns an iterator that produces consecutive integers, indefinitely. The first number can be passed as an argument (the default is zero). There is no upper bound argument.

The `cycle()` function returns an iterator that repeats the contents of the arguments it is given indefinitely. Because it has to remember the entire contents of the input iterator, it may consume quite a bit of memory if the iterator is long.

The `repeat()` function returns an iterator that produces the same value each time it is accessed.

1.2.4 过滤

The `dropwhile()` function returns an iterator that produces elements of the input iterator after a condition becomes false for the first time.

The opposite of `dropwhile()` is `takewhile()`. It returns an iterator that itself returns items from the input iterator as long as the test function returns true.

The built-in function `filter()` returns an iterator that includes only items for which the test function returns true.

`filterfalse()` returns an iterator that includes only items where the test function returns false.

`compress()` offers another way to filter the contents of an iterable. Instead of calling a function, it uses the values in another iterable to indicate when to accept a value and when to ignore it.

1.2.5 数据分组

The `groupby()` function returns an iterator that produces sets of values organized by a common key. 输入的序列要根据键值排序，以保证得到预期的分组。

1.2.6 合并输入

The `accumulate()` function processes the input iterable, passing the n th and $n + 1$ st item to a function and producing the return value instead of either input. The default function used to combine the two values adds them, so `accumulate()` can be used to produce the cumulative sum of a series of numerical inputs. `accumulate()` 可以与任何取两个输入值得函数结合来得到不同得结果。

迭代处理多个序列的嵌套 `for` 循环通常可以被替换为 `product()`，它会生成一个迭代器，值为输入值集合的笛卡尔积。The values produced by `product()` are tuples, with the members taken from each of the iterables passed in as arguments in the order they are passed. The first tuple returned includes the first value from each iterable. The last iterable passed to `product()` is processed first, followed by the next-to-last, and so on.

The `permutations()` function produces items from the input iterable combined in the possible permutations of the given length. It defaults to producing the full set of all permutations.

为了将值限制为唯一的组合而不是排列，可以使用 `combinations()`。只要输入的成员是唯一的，输出就不会包含任何重复的值。

1.3 operator: 内置操作符的函数接口

Programming using iterators occasionally requires creating small functions for simple expressions. Sometimes, these can be implemented as lambda functions, but for some operations new functions are not needed at all. The `operator` module defines functions that correspond to the built-in arithmetic, comparison, and other operations for the standard object APIs.

1.3.1 逻辑操作**1.3.2 比较操作符****1.3.3 算术操作符****1.3.4 序列操作符**

The operators for working with sequences can be organized into four groups: building up sequences, searching for items, accessing contents, and removing items from sequences.

1.3.5 就地操作符**1.3.6 属性和元素“获取方法”****1.3.7 结合操作符和定制类**