Python3 标准库

Stephen CUI

2023-09-20

Contents

Chapter 1

文本

1.1 re: 正则表达式

1.1.1 模式语法

重复

模式中有 5 中表示重复的方法

1. 元字符 *，则表示重复 0 次或多次（运行一个模式重复 0 次是指这个模式即使不出现也可以出现匹配）
2. + ，那么模式必须至少出现 1 次才能匹配
3. ? 表示出现 0 次或 1 次
4. 如果需要指定出现次数，需要在模式后面使用 $\{m\}$，这里 m 是模式应重复的次数
5. 如果要允许一个可变但有限的重复次数，那么可以使用 $\{m,n\}$，这里 m 是最小重复次数，n 是最大重复次数。如果省略 n (m,) 则表示值必须知道出现 m 次，但没有最大限制

字符集

A character set is a group of characters, any one of which can match at that point in the pattern. For example, [ab] would match either a or b.

A character set can also be used to exclude specific characters. The carat ^ means to look for characters that are not in the set following the carat.

As character sets grow larger, typing every character that should (or should not) match becomes tedious. A more compact format using **character ranges** can be used to define a character set to include all of the contiguous characters between the specified start and stop points.

As a special case of a character set, the meta-character dot, or period (.), indicates that the pattern should match any single character in that position.

转义码

锚定

使用锚定指令指定模式在输入文本中的相对位置。

Table 1.1: Regular Expression Escape Codes

Code	Meaning
d	A digit
D	A non-digit
s	Whitespace (tab, space, newline, etc.)
S	Non-whitespace
w	Alphanumeric
W	Non-alphanumeric

Table 1.2: 正则表达式锚定码

锚定码	含义
^	字符串或行的开头
$	字符串或行末尾
A	字符串开头
Z	字符串末尾
b	单词开头或末尾的空串
B	不在单词开头或末尾的空串

Chapter 2

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 包含的函数用你与创建函数修饰符、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。

2.1 functools: 管理函数的工具

2.1.1 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()`).

2.1.2 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.

2.1.3 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。

2.2 itertools: 迭代器函数

itertools 包括一组用于处理序列数据集的函数。

2.2.1 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。**

2.2.2 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).

2.2.3 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.

2.2.4 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.

2.2.5 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。

2.2.6 合并输入

The accumulate() function processes the input iterable, passing the nth 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。

2.3 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.

2.3.1 逻辑操作

2.3.2 比较操作符

2.3.3 算术操作符

2.3.4 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.

2.3.5 就地操作符

2.3.6 属性和元素“获取方法”

2.3.7 结合操作符和定制类

Chapter 3

日期和时间

不同于 int、float 和 str，Python 没有包含对应日期和时间的原生类型，不过提供了 3 个相应的模块，可以采用多种表示来管理日期和时间值。

time 模块由底层 C 库提供与事件相关的函数。它包含一些函数，可以用于获取时钟时间和处理器运行时间，还提供了多种表示来管理日期和时间值。

datetime 模块为日期、时间以及日期时间值提供了一个更高层接口。datetime 中的类支持算术、比较和时区设置。

calendar 模块可以创建周、月和年的格式化表示。它还可以用来计算重复事件，给定日期是星期几，以及其他基于日历的值。

3.1 time: 时钟日期

The time module provides access to several types of clocks, each useful for different purposes. The standard system calls such as `time()` report the system "wall clock" time. The `monotonic()` clock can be used to measure elapsed time in a long-running process because it is guaranteed never to move backward, even if the system time is changed. For performance testing, `perf_counter()` provides access to the clock with the highest available resolution, which makes short time measurements more accurate. The CPU time is available through `clock()`, and `process_time()` returns the combined processor time and system time.

3.2 datetime: 日期和时间值管理

3.3 calendar: 处理日期

Chapter 4

应用构建模块

4.1 logging: 报告状态、错误和信息消息

4.1.1 日志系统的组成

The logging system consists of four interacting types of objects. Each module or application that wants to log some activity uses a Logger instance to add information to the logs. Invoking the logger creates a LogRecord, which holds the information in memory until it is processed. A Logger may have a number of Handler objects configured to receive and process log records. The Handler uses a Formatter to turn the log records into output messages.

4.2 configparser: 处理配置文件

Use the configparser module to manage user-editable configuration files for an application using a format similar to Windows INI files. The contents of the configuration files can be organized into groups and several option value types are supported, including integers, floating-point values, and booleans. Option values can be combined using Python formatting strings to build longer values such as URLs from shorter values such as host names and port numbers.

Table 4.1: 日志级别

级别	值
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
UNSET	0

4.2.1 配置文件格式

The file format used by configparser is similar to the format used by older versions of Microsoft Windows. It consists of one or more named sections, each of which can contain individual options with names and values.

The parser identifies config file sections by looking for lines starting with [and ending with]. The value between the square brackets is the section name, and can contain any characters except square brackets.

Options are listed one per line within a section. The line starts with the name of the option, which is separated from the value by a colon (:) or equal sign (=). Whitespace around the separator is ignored when the file is parsed.

Lines starting with a semicolon (;) or an octothorpe (#) are treated as comments. They are ignored when the contents of the configuration file are accessed programmatically.

读取配置文件

可以使用 Configparser 的 read 方法来读取配置文件。

read 方法还接受一个文件名列表。依次检查这个列表中的各个名，如果文件存在，就打开并读取该文件。

read 返回一个列表，其中包含成功加载的文件的名。通过检查这个列表，程序可以发现缺少哪些配置文件，并确定是将其忽略还是把这个条件当作一个错误。

包含 Unicode 数据的配置文件应当使用适当的编码值来读取。

4.2.2 访问配置设置

ConfigParser 包含一些方法来检查所解析配置的结构，包括列出节和选项，以及得到它们的值。

sections() 和 options() 会返回字符串列表，而 items() 返回一个元组列表，元组包含名 - 值对。

ConfigParser 还支持与 dict 同样的映射 API，ConfigParser 相当于一个字典，其中包含对应各个节的不同字典。

测试值是否存在 要测试一个节是否存在，可以使用 `has_section()`，并传入节名作为方法参数。在调用 get() 之前先测试一个节是否存在，这样可以避免因为缺少数据而导致产生异常。

`has_option()` 可以测试一个节中某个选项是否存在。如果节不存在，那么 `has_option()`会返回 False。

值类型 所有节和选项名都被处理为字符串，不过选项值可以是字符串、整数、浮点数或者布尔值。可以用多个不同的字符串值表示配置文件中的布尔值；访问时它们会被自动转换为 True 或 False。ConfigParser 不会尝试去了解选项类型，而会希望应用使用正确的方法来获取所需类型的值。get() 总会返回一个字符串。使用 getint 可以得到整数，getfloat 得到浮点数，使用 getboolean 得到布尔值。

可以在 ConfigParser 的 converters 参数中传入转换函数来增加定制类型转换器。每个转化器接受一个输入值，然后将他转换为适当的返回类型。增加转换器会让 ConfigParser 自动为这个类型创建一个获取方法，并使用 converters 中指定的类型名。还可以向 ConfigParser 的子类直接增加转换器方法。

选项作为标志 通常，解析器要求每个选项都有一个明确的值，不过如果 ConfigParser 参数 `allow_no_value` 被设置为 True，那么选项可以在输入文件中单独作为一行，而且还可以被用作一个标志。选项没有明确的值时，`get_option()`会报告这个选项存在，并且 get() 返回 None。

多行字符串 字符串值可以跨多行，前提是后面的行要缩进。在缩进的多行值中，空行会作为值得一部分保留。

4.2.3 修改设置

ConfigParser 主要通过从文件读取设置来进行配置，不过也可以填充设置，通过调用 `add_section()` 来创建一个新的节，另外调用 set() 可以增加或修改一个选项。所有选项都被设置为字符串，即使它们将被获取为整数、浮点数或布尔值。

可以分别用 `remove_section()` 和 `remove_option()` 从 ConfigParser 删除节和选项。删除一节也会删除其中包含得所有选项。

4.2.4 保存配置文件

用所需得数据填充 ConfigParser 后，就可以调用 write() 方法将它保存到一个文件。这种方法可以用来提供一个用于编辑设置得用户接口，而不需要编写任何代码来管理文件。

> **警告**
>
> 读取、修改和重写配置文件时，原配置文件中的注释不会保留。

4.2.5 选项搜索路径

ConfigParser 查找选项时使用了一个多步搜索过程。开始搜索选项之前，首先会测试节点。如果这个节不存在，而且名不是特殊值 DEFAULT，则产生一个 NoSectionError 异常。

1. 如果选项名出现在传递到 get() 的 vars 字典中，则会返回 vars 的值；
2. 如果选项名出现在指定的节中，则返回该节中的值；
3. 如果选项名出现在 DEFAULT 节中，则会返回相应的值；
4. 如果选项名出现在传递到构造函数的 defaults 字典中，则会返回相应的值；

如果这个名未出现在以上任何位置，则产生 NoOptionError。

4.2.6 用拼接合并值

ConfigParser 提供了一个特性，名为拼接，可以将值结合在一起。如果值包含标准 Python 格式串，那么获取这个值时就会触发拼接特性。获取的值中指定的各个选项会按顺序一次被替换为相应的值，直到不再需要更多替换。

使用默认值 并不要求拼接的值出现在原选项所在的同一节中。默认值可以与覆盖值混合使用。

替换错误 `MAX_INTERPOLATION_DEPTH` 步骤之后替换停止，以避免递归引用导致的问题。如果有过多替换步骤，则会产生一个 InterpolationDepthError 异常。缺少值会导致一个 InterpolationMissingOptionError 异常。

转义特殊字符 由于拼接字符以 % 开始，值中的字面量 % 必须转义为 %%。

拓展拼接 ConfigParser 通过 interpolation 参数来支持候选的拼接实现。interpolation 参数给定的对象要实现 Interpolation 类定义的 API。

禁用拼接 如果要禁用拼接，则应传入 None 而不是 Interpolation 对象。

Chapter 5

5.1 platform: 系统版本信息

尽管 Python 通常被用作一个跨平台的语言，但是有时还是有必要知道程序在哪种系统上运行。构建工具需要这个信息，另外应用可能也需要直到它使用的一些库或外部命令在不同操作系统上有不同的接口。

5.1.1 解释器

`python_version()`和`python_version_tuple()`可以返回不同形式的解释器版本，包括主版本、次版本和补丁级组件。`python_compiler()`会报告构建解释器所用的编译器。`python_build()` 将给出解释器构建的版本串。

5.1.2 平台

platform() 函数返回一个字符串，其中包含一个通用的平台标识符。这个函数接受两个可选的布尔参数。如果 aliased 为 True，则返回值中的名为会从一个正式名转换为更常用的格式。如果 terse 为 True，则会返回一个最小值，即取出某些部分，而不是返回完整的串。

5.1.3 操作系统和硬件信息

uname() 返回一个元组，其中包含系统、节点、发行号、版本、机器和处理器值。可以通过同名的函数访问各个值，比如表所列。

Table 5.1: 平台信息函数

Function	Return Value
system()	操作系统名
node()	服务器主机名，不是完全限定名
release()	操作系统发行号
version()	更详细的系统版本信息
machine()	硬件类型标识符，如 'i386'
processor()	处理器实际标识符（有些情况下与 machine() 值相同）

5.1.4 可执行程序体系结构

可以使用 architecture() 函数查看程序的体系结构信息。第一个参数是可执行程序的路径（默认为 sys.executable，即 Python 解释器）。返回值是一个元组，包含位体系结构和使用的链接格式）。