ipaddress 模組介紹

發🄵 *3.9.2*

Guido van Rossum
and the Python development team

4 月 02, 2021

Python Software Foundation
Email: docs@python.org

Contents

作者 Peter Moody

作者 Nick Coghlan

概述

本文档旨在简要介绍 `ipaddress` 模块。它主要针对那些不熟悉 IP 网络术语的用户，但也可能对想要速览 `ipaddress` 如何代表 IP 网络寻址概念的网络工程师有用。

1 创建 Address/Network/Interface 对象

因为 ipaddress 是一个用于检查和操作 IP 地址的模块，你要做的第一件事就是创建一些对象。您可以使用 ipaddress 从字符串和整数创建对象。

1.1 关于 IP 版本的说明

对于不太熟悉 IP 寻址的读者，重要的是要知道 Internet 协议当前正在从协议的版本 4 转移到版本 6。转换很大程度上是因为协议的版本 4 没有提供足够的地址来满足整个世界的需求，特别是考虑到越来越多的设备直接连接到互联网。

解释协议的两个版本之间的差异的细节超出了本介绍的范围，但读者需要至少知道存在这两个版本，并且有时需要强制使用一个版本或其他版本。

1.2 IP 主机地址

通常称为“主机地址”的地址是使用 IP 寻址时最基本的单元。创建地址的最简单方法是使用 ipaddress.ip_address() 工厂函数，该函数根据传入的值自动确定是创建 IPv4 还是 IPv6 地址：

```
>>> ipaddress.ip_address('192.0.2.1')
IPv4Address('192.0.2.1')
>>> ipaddress.ip_address('2001:DB8::1')
IPv6Address('2001:db8::1')
```

地址也可以直接从整数创建，适配 32 位的值并假定为 IPv4 地址:

```
>>> ipaddress.ip_address(3221225985)
IPv4Address('192.0.2.1')
>>> ipaddress.ip_address(42540766411282592856903984951653826561)
IPv6Address('2001:db8::1')
```

要强制使用 IPv4 或 IPv6 地址，可以直接调用相关的类。这对于强制为小整数创建 IPv6 地址特别有用:

```
>>> ipaddress.ip_address(1)
IPv4Address('0.0.0.1')
>>> ipaddress.IPv4Address(1)
IPv4Address('0.0.0.1')
>>> ipaddress.IPv6Address(1)
IPv6Address('::1')
```

1.3 定义网络

主机地址通常组合在一起形成 IP 网络，因此 ipaddress 提供了一种创建、检查和操作网络定义的方法。IP 网络对象由字符串构成，这些字符串定义作为该网络一部分的主机地址范围。该信息的最简单形式是“网络地址/网络前缀”对，其中前缀定义了比较的前导比特数，以确定地址是否是网络的一部分，并且网络地址定义了那些位的预期值。

对于地址，提供了一个自动确定正确 IP 版本的工厂函数:

```
>>> ipaddress.ip_network('192.0.2.0/24')
IPv4Network('192.0.2.0/24')
>>> ipaddress.ip_network('2001:db8::0/96')
IPv6Network('2001:db8::/96')
```

网络对象不能设置任何主机位。这样做的实际效果是 “192.0.2.1/24“没有描述网络。这种定义被称为接口对象，因为网络上 IP 表示法通常用于描述给定网络上的计算机的网络接口，并在下一节中进一步描述。

默认情况下，尝试创建一个设置了主机位的网络对象将导致 `ValueError` 被引发。要请求将附加位强制为零，可以将标志 “strict=False“ 传递给构造函数:

```
>>> ipaddress.ip_network('192.0.2.1/24')
Traceback (most recent call last):
  ...
ValueError: 192.0.2.1/24 has host bits set
>>> ipaddress.ip_network('192.0.2.1/24', strict=False)
IPv4Network('192.0.2.0/24')
```

虽然字符串形式提供了更大的灵活性，但网络也可以用整数定义，就像主机地址一样。在这种情况下，网络被认为只包含由整数标识的单个地址，因此网络前缀包括整个网络地址:

```
>>> ipaddress.ip_network(3221225984)
IPv4Network('192.0.2.0/32')
>>> ipaddress.ip_network(42540766411282592856903984951653826560)
IPv6Network('2001:db8::/128')
```

与地址一样，可以通过直接调用类构造函数而不是使用工厂函数来强制创建特定类型的网络。

1.4 主机接口

如上所述，如果您需要描述特定网络上的地址，则地址和网络类都不够。像 `192.0.2.1/24` 这样的表示法通常被网络工程师和为防火墙和路由器编写工具的人用作 “`192.0.2.0/24` 网络上的主机 `192.0.2.1` ” 的简写。因此，`ipaddress` 提供了一组将地址与特定网络相关联的混合类。用于创建的接口与用于定义网络对象的接口相同，除了地址部分不限于是网络地址。

```
>>> ipaddress.ip_interface('192.0.2.1/24')
IPv4Interface('192.0.2.1/24')
>>> ipaddress.ip_interface('2001:db8::1/96')
IPv6Interface('2001:db8::1/96')
```

接受整数输入（与网络一样），并且可以通过直接调用相关构造函数来强制使用特定 IP 版本。

2 审查 Address/Network/Interface 对象

你已经遇到了创建 IPv(4|6)(Address|Network|Interface) 对象的麻烦，因此你可能希望获得有关它的信息。`ipaddress` 试图让这个过程变得简单直观。

提取 IP 版本:

```
>>> addr4 = ipaddress.ip_address('192.0.2.1')
>>> addr6 = ipaddress.ip_address('2001:db8::1')
>>> addr6.version
6
>>> addr4.version
4
```

从接口获取网络:

```
>>> host4 = ipaddress.ip_interface('192.0.2.1/24')
>>> host4.network
IPv4Network('192.0.2.0/24')
>>> host6 = ipaddress.ip_interface('2001:db8::1/96')
>>> host6.network
IPv6Network('2001:db8::/96')
```

找出网络中有多少独立地址:

```
>>> net4 = ipaddress.ip_network('192.0.2.0/24')
>>> net4.num_addresses
256
>>> net6 = ipaddress.ip_network('2001:db8::0/96')
>>> net6.num_addresses
4294967296
```

迭代网络上的“可用”地址:

```
>>> net4 = ipaddress.ip_network('192.0.2.0/24')
>>> for x in net4.hosts():
...     print(x)
192.0.2.1
192.0.2.2
192.0.2.3
192.0.2.4
...
192.0.2.252
192.0.2.253
192.0.2.254
```

获取网络掩码（即对应于网络前缀的设置位）或主机掩码（不属于网络掩码的任何位）:

```
>>> net4 = ipaddress.ip_network('192.0.2.0/24')
>>> net4.netmask
IPv4Address('255.255.255.0')
>>> net4.hostmask
IPv4Address('0.0.0.255')
>>> net6 = ipaddress.ip_network('2001:db8::0/96')
>>> net6.netmask
IPv6Address('ffff:ffff:ffff:ffff:ffff:ffff::')
>>> net6.hostmask
IPv6Address('::ffff:ffff')
```

展开或压缩地址:

```
>>> addr6.exploded
'2001:0db8:0000:0000:0000:0000:0000:0001'
>>> addr6.compressed
'2001:db8::1'
>>> net6.exploded
'2001:0db8:0000:0000:0000:0000:0000:0000/96'
>>> net6.compressed
'2001:db8::/96'
```

虽然 IPv4 不支持展开或压缩，但关联对象仍提供相关属性，因此版本中性代码可以轻松确保最简洁或最详细的形式用于 IPv6 地址，同时仍能正确处理 IPv4 地址。

3 Network 作为 Address 列表

将网络视为列表有时很有用。这意味着它可以像这样索引它们:

```
>>> net4[1]
IPv4Address('192.0.2.1')
>>> net4[-1]
IPv4Address('192.0.2.255')
>>> net6[1]
IPv6Address('2001:db8::1')
>>> net6[-1]
IPv6Address('2001:db8::ffff:ffff')
```

它还意味着网络对象可以使用像这样的列表成员测试语法:

```
if address in network:
    # do something
```

根据网络前缀有效地完成包含性测试:

```
>>> addr4 = ipaddress.ip_address('192.0.2.1')
>>> addr4 in ipaddress.ip_network('192.0.2.0/24')
True
>>> addr4 in ipaddress.ip_network('192.0.3.0/24')
False
```

4 比较

ipaddress 有意义地提供了一些简单、希望直观的比较对象的方法:

```
>>> ipaddress.ip_address('192.0.2.1') < ipaddress.ip_address('192.0.2.2')
True
```

如果你尝试比较不同版本或不同类型的对象，则会引发 TypeError 异常。

5 将 IP 地址与其他模块一起使用

其他使用 IP 地址的模块（例如 socket ）通常不会直接接受来自该模块的对象。相反，它们必须被强制转换为另一个模块可接受的整数或字符串:

```
>>> addr4 = ipaddress.ip_address('192.0.2.1')
>>> str(addr4)
'192.0.2.1'
>>> int(addr4)
3221225985
```

6 实例创建失败时获取更多详细信息

使用与版本无关的工厂函数创建 address/network/interface 对象时，任何错误都将报告为 `ValueError`，带有一般错误消息，只是说传入的值未被识别为该类型的对象。缺少特定错误是因为有必要知道该值是 * 假设 * 是 IPv4 还是 IPv6，以便提供有关其被拒绝原因的更多详细信息。

为了支持访问这些额外细节的用例，各个类构造函数实际上引发了 `ValueError` 子类 `ipaddress.AddressValueError` 和 `ipaddress.NetmaskValueError` 以准确指示定义的哪一部分无法正确解析。

直接使用类构造函数时，错误消息更加详细。例如:

```
>>> ipaddress.ip_address("192.168.0.256")
Traceback (most recent call last):
  ...
ValueError: '192.168.0.256' does not appear to be an IPv4 or IPv6 address
>>> ipaddress.IPv4Address("192.168.0.256")
Traceback (most recent call last):
  ...
ipaddress.AddressValueError: Octet 256 (> 255) not permitted in '192.168.0.256'

>>> ipaddress.ip_network("192.168.0.1/64")
Traceback (most recent call last):
  ...
ValueError: '192.168.0.1/64' does not appear to be an IPv4 or IPv6 network
>>> ipaddress.IPv4Network("192.168.0.1/64")
Traceback (most recent call last):
  ...
ipaddress.NetmaskValueError: '64' is not a valid netmask
```

但是，两个模块特定的异常都有 `ValueError` 作为它们的父类，所以如果你不关心特定类型的错误，你仍然可以编写如下代码:

```
try:
    network = ipaddress.IPv4Network(address)
except ValueError:
    print('address/netmask is invalid for IPv4:', address)
```