eFusion 07 августа 2020

Концепция IP-адресов на примере Python-модуля ipaddress

Рассказываем, как работать с IP-адресами классического протокола IPv4 в теории и на практике – в коде Python. Показываем, как проверить связь между группой IP-адресов и их соответствие подсетям и диапазонам частных IP-адресов.

Вводные сведения об IP-адресах

```
220.14.9.37
```

Каждый октет – это один байт, число от 0 до 255. То есть максимальный адрес равен `255.255.255.255`, а минимальный – `0.0.0.0`.

Далее мы рассмотрим, как модуль `ipaddress` выполняет преобразования адреса так, что нам не приходится отвлекаться на строение адреса.

Примечание

Модуль ipaddress добавлен в стандартную библиотеку в версии Python 3.3, примеры туториала проверены на примере Python 3.8. Пользователь matyushkin любезно адаптировал программный код статьи в виде конспективного Jupyter-блокнота. За счёт этого с кодом можно поиграть в среде Colab.

Модуль ipaddress

Получим внешний IP-адрес нашего компьютера для работы с ним в командной строке. В Linux это делается так:

```
$ curl -sS ifconfig.me/ip
220.14.9.37
```

Этот запрос узнает наш IP-шник на сайте ifconfig.me. Сайт также выдает множество другой полезной информации о вашем сетевом подключении.

Примечания

Возможно, запрос вернет не лично ваш внешний (реальный) адрес. Если соединение находится за NAT-ом, то этот адрес используют и другие абоненты, находящиеся в вашем сегменте сети. Точную информацию о том, как реализована сеть, можно узнать у ее провайдера.

Теперь откроем интерпретатор Python. Чтобы создать объект Python с инкапсулированным адресом, создадим класс `IPv4Address`:

```
>>> from ipaddress import IPv4Address
```

Передача строки `"220.14.9.37"` в конструктор `IPv4Address` – наиболее распространенный подход, но класс может принимать и другие типы:

```
>>> IPv4Address(3691907365)   # Из целого числа
IPv4Address('220.14.9.37')

>>> IPv4Address(b"\xdc\x0e\t%")  # Из байтовой строки
IPv4Address('220.14.9.37')
```

Адрес можно распаковать в требуемую форму:

```
>>> int(addr)
3691907365
>>> addr.packed
b'\xdc\x0e\t%'
```

Экземпляры `IPv4Address` являются хэшируемыми и могут использоваться в качестве ключей словаря:

```
>>> hash(IPv4Address("220.14.9.37"))
4035855712965130587

>>> num_connections = {
...     IPv4Address("220.14.9.37"): 2,
...     IPv4Address("100.201.0.4"): 16,
...     IPv4Address("8.240.12.2"): 4,
... }
```

Класс `IPv4Address` также реализует методы, позволяющие проводить сравнения:

```
>>> IPv4Address("220.14.9.37") > IPv4Address("8.240.12.2")
True

>>> addrs = (
...     IPv4Address("220.14.9.37"),
```

```
...     print(a)
...
8.240.12.2
100.201.0.4
220.14.9.37
```

Можно использовать любой стандартный оператор сравнения целочисленных значений адресных объектов.

IP-сети и интерфейсы

Сеть – это набор IP-адресов. Сети описываются и отображаются как непрерывные диапазоны адресов. Например, сеть может соответствовать диапазону `192.4.2.0 – 192.4.2.255`, т. е. включать 256 адресов. Если нужно это отобразить в краткой форме, используется нотация CIDR.

В CIDR сеть определяется с помощью сетевого адреса и префикса `<network_address>/<prefix>`:

```
>>> from ipaddress import IPv4Network
>>> net = IPv4Network("192.4.2.0/24")
>>> net.num_addresses
256

# Вывести префикс можно с помощью свойства prefixlen:
>>> net.prefixlen
24
```

В данном случае префикс равен 24. Префикс – это количество ведущих битов, соответствующих входящим в сеть адресам. Ведущие биты отсчитываются слева направо.

Пример: входит ли адрес 192.4.2.12 в сеть 192.4.2.0/24?

Ответ: да, так как ведущие 24 бита адреса `192.4.2.12` – это первые три октета: `192.4.2`. Последний октет соответствует последним 8 битам 32-битного IP-адреса.

Воспользуемся `netmask` для маскирования битов в сравниваемых адресах.

Определение

Битовая маска – данные, которые используются для **маскирования** – выбора отдельных битов

```
>>> net.netmask
IPv4Address('255.255.255.0')
```

На рисунке ниже показано, как сравниваются ведущие биты, чтобы определить, является ли адрес частью сети.

```
11000000 00000100 00000010 00001100  # 192.4.2.12  # Host IP address
11000000 00000100 00000010 00000000  # 192.4.2.0   # Network address
                           |
                           ^ 24th bit (stop here!)
|__________________________|
           |
     These bits match
```

Побитовое сравнение

Последние 8 бит в `192.4.2.12` маскируются нулем и игнорируются при сравнении.

```
>>> IPv4Address("192.4.2.12") in net
True
>>> IPv4Address("192.4.20.2") in net
False
```

Рассмотрим еще один важный тип адреса – широковещательный.

Определение

Широковещательный адрес – условный (не присвоенный никакому устройству в сети) адрес, который используется для передачи широковещательных пакетов в компьютерных сетях.

Это единственный адрес, который может использоваться для связи со всеми хостами сети:

```
>>> net.network_address
IPv4Address('192.4.2.0')
```

Чаще всего вы будете сталкиваться с длиной префикса кратной 8.

Prefix Length	Number of Addresses	Netmask
8	16,777,216	255.0.0.0
16	65,536	255.255.0.0
24	256	255.255.255.0
32	1	255.255.255.255

Распространенные подсети

Любое целое число от 0 до 32 является допустимым, но такой вариант встречается реже:

```
>>> net = IPv4Network("100.64.0.0/10")
>>> net.num_addresses
4194304
>>> net.netmask
IPv4Address('255.192.0.0')
```

Перебор IP-адресов в цикле

Класс `IPv4Network` позволяет перебирать отдельные адреса в цикле for:

```
>>> net = IPv4Network("192.4.2.0/28")
>>> for addr in net:
...     print(addr)
...
192.4.2.0
192.4.2.1
192.4.2.2
...
192.4.2.13
192.4.2.14
192.4.2.15
```

```
>>> h = net.hosts()
>>> type(h)
<class 'generator'>
>>> next(h)
IPv4Address('192.4.2.1')
>>> next(h)
IPv4Address('192.4.2.2')
```

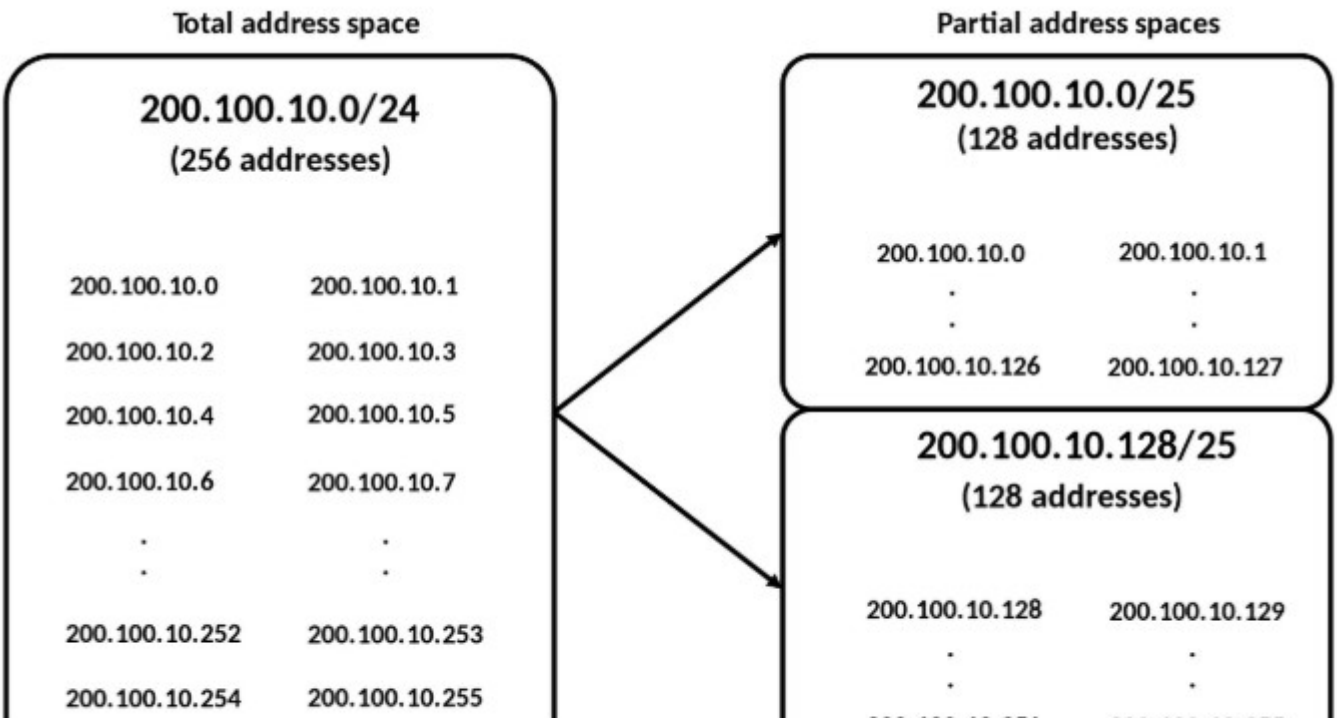
Подсети IP-адресов

Подсеть – это часть IP-сети:

```
>>> small_net = IPv4Network("192.0.2.0/28")
>>> big_net = IPv4Network("192.0.0.0/16")
>>> small_net.subnet_of(big_net)
True
>>> big_net.supernet_of(small_net)
True
```

В коде выше `small_net` содержит 16 адресов, а `big_net` – 65 536.

Распространенный способ разбиения на подсети – это увеличение длины префикса на 1:



К счастью, `IPv4Network` расчеты подсетей поддерживаются встроенным методом `subnets()`:

```
>>> for sn in net.subnets():
...     print(sn)
...
200.100.10.0/25
200.100.10.128/25
```

В передаваемом `subnets()` аргументе можно задать, каким должен быть новый префикс:

```
>>> for sn in net.subnets(new_prefix=28):
...     print(sn)
...
200.100.10.0/28
200.100.10.16/28
200.100.10.32/28
...
200.100.10.208/28
200.100.10.224/28
200.100.10.240/28
```

Специальные диапазоны IP-адресов

Администрация адресного пространства Интернет (Internet Assigned Numbers Authority, IANA) совместно с Инженерном советом Интернета (Internet Engineering Task Force, IETF) осуществляют надзор за распределением диапазонов адресов. Реестр подобных адресов – важная таблица, которая описывает, для каких целей зарезервированы диапазоны IPv4-адресов.

К примеру, это частные IP-адреса, используемые для внутренней связи между устройствами в сети, не требующей подключения к интернету:

Range	Number of Addresses	Network Address	Broadcast Address
10.0.0.0/8	16,777,216	10.0.0.0	10.255.255.255
172.16.0.0/12	1,048,576	172.16.0.0	172.31.255.255

Случайным образом выберем адрес – `10.243.156.214`. Относится ли этот адрес к приватным? Для этого проверим, попадает ли он в диапазон сети `10.0.0.0/8`:

```
>>> IPv4Address("10.243.156.214") in IPv4Network("10.0.0.0/8")
True
```

Другой специальный тип адреса – это локальный адрес связи, состоящий из блока `169.254.0.0/16`. Примером может служить Amazon Time Sync Service, доступный для инстансов AWS EC2 по адресу `169.254.169.123`. Данный пул также использует Windows для выдачи адресов сетевым адаптерам при отсутствии интернета от провайдера.

```
>>> timesync_addr = IPv4Address("169.254.169.123")
>>> timesync_addr.is_link_local
True
```

Модуль `ipaddress` предоставляет набор свойств для проверки того, относится ли адрес к специальным:

```
>>> IPv4Address("10.243.156.214").is_private
True
>>> IPv4Address("127.0.0.1").is_loopback
True

>>> [i for i in dir(IPv4Address) if i.startswith("is_")]
['is_global',
 'is_link_local',
 'is_loopback',
 'is_multicast',
 'is_private',
 'is_reserved',
 'is_unspecified']
```

Вот еще несколько зарезервированных сетей:

- `0.0.0.0/8` – адреса источников пакетов «своей» сети;

Что происходит внутри ipaddress

В дополнение к хорошо документированному API, исходный код CPython и класс IPv4Address показывают некоторые отличные идеи, как улучшить собственный код.

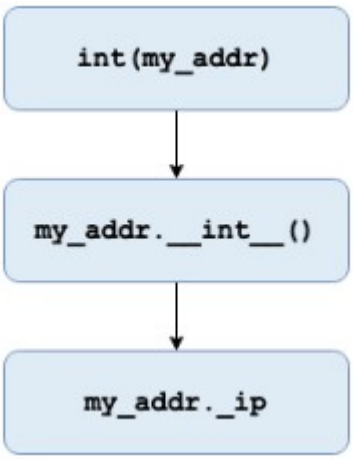
Компоновщик

Модуль `ipaddress` использует преимущества шаблона проектирования «Компоновщик». Класс `IPv4Address` представляет собой компоновщик, который оборачивает обычное целое число.

Каждый экземпляр `IPv4Address` имеет атрибут **`_ip`**, число типа `int`. Многие свойства и методы класса определяются значением этого атрибута:

```
>>> addr = IPv4Address("220.14.9.37")
>>> addr._ip
3691907365
```

Атрибут `_ip` отвечает за создание `int(addr)`. Цепочка вызовов выглядит следующим образом:



Цепочка вызовов в компоновщике

Продемонстрируем силу `._ip` путем расширения класса `IPv4Address`:

```
from ipaddress import IPv4Address

class MyIPv4(IPv4Address):
    def __and__(self, other: IPv4Address):
        if not isinstance(other, (int, IPv4Address)):
```

Добавление .__and__() позволяет использовать бинарный оператор `&`, чтобы применять маску к IP-адресу:

```
>>> addr = MyIPv4("100.127.40.32")
>>> mask = MyIPv4("255.192.0.0")  # Соответствует префиксу /10

>>> addr & mask
MyIPv4('100.64.0.0')

>>> addr & 0xffc00000  # hex-литерал для 255.192.0.0
MyIPv4('100.64.0.0')
```

Метод `__and__()` позволяет использовать либо другой IPv4Address, либо непосредственно int в качестве маски. Поскольку MyIPv4 является подклассом IPv4Address, проверка `isinstance()` в данном случае вернет `True`.

Помимо перегрузки оператора, есть возможность добавить новые свойства:

```
import re
from ipaddress import IPv4Address

class MyIPv4(IPv4Address):
    @property
    def binary_repr(self, sep=".") -> str:
        """Представляет IPv4 в виде 4 блоков по 8 бит."""
        return sep.join(f"{i:08b}" for i in self.packed)  # 8 строка

    @classmethod
    def from_binary_repr(cls, binary_repr: str):
        """Создает IPv4 из двоичного представления."""
        i = int(re.sub(r"[^01]", "", binary_repr), 2)  # 14 строка
        return cls(i)
```

В методе `binary_repr` (строка 8), используется `.packed` для преобразования IP-адреса в массив байтов, который затем форматируется, как строковое представление бинарной формы.

В `from_binary_repr`, вызов `int(re.sub(r"[^01]", "", binary_repr), 2)` (строка 14) состоит

- анализ результата с помощью `int(<string>, 2)`.

Методы `binary_repr()` и `from_binary_repr()` позволяют проводить двустороннюю конвертацию:

```
>>> MyIPv4("220.14.9.37").binary_repr
'11011100.00001110.00001001.00100101'
>>> MyIPv4("255.255.0.0").binary_repr  # Маска для префикса /16
'11111111.11111111.00000000.00000000'
```


Таким образом, мы разобрали несколько способов использования преимуществ шаблона IP-as-integer, который может помочь расширить функциональность IPv4Address с небольшим количеством дополнительного кода.

Заключение

Если вам нравится язык Python и вы хотите детально овладеть стандартной библиотекой, у нас есть множество родственных публикаций:

- Как хранить объекты Python со сложной структурой (о модуле pickle)
- Итерируем правильно: 20 приемов использования в Python модуля itertools
- Не изобретать велосипед, или Обзор модуля collections в Python
- Назад в будущее: практическое руководство по путешествию во времени с Python
- Как подружить Python и базы данных SQL. Подробное руководство

Источники

- https://realpython.com/python-ipaddress-module/

30 сентября Онлайн Бесплатно

VTB API hackathon 2023

25 сентября Москва Онлайн Бесплатно

Приглашаем аналитиков присоединиться к проекту IT_ONE CAREER и испытать себя на онлайн-хакатоне!

29 сентября Онлайн Бесплатно

+ Показать еще

Комментарии

Оставьте свой комментарий (можно использовать markdown)

Отправить

Популярные По порядку

Леонид Хозяинов 27 ноября 2020 0 1

Введение в модуль ipaddress на русском https://digitology.tech/docs/python_3/howto/ipaddress.html

Перевод официальной документации по библиотеке ipaddress
https://digitology.tech/docs/python_3/howto/ipaddress.html

Ответить •••

ВАКАНСИИ Добавить вакансию

Data Scientist (стажер)
Москва, по итогам собеседования

Контент-менеджер
по итогам собеседования

ЛУЧШИЕ СТАТЬИ ПО ТЕМЕ

ООП на Python: концепции, принципы и примеры реализации

Программирование на Python допускает различные методологии, но в его основе лежит объектный подход, поэтому работать в стиле ООП на Python очень просто.

8 книг по компьютерным сетям

Подборка актуальных книг по современным сетям, где каждый — от новичка до профессионала —