tw_community сегодня в 13:16

Развлечения с парсингом IP-адресов

Автор оригинала: David Anderson

Блог компании Timeweb, Ненормальное программирование, Программирование

Перевод

Решив заняться созданием быстрого парсера IPv4+6, я написал медленный, но правильный парсер, который можно было бы использовать как базу для сравнения. В процессе его создания я узнал множество ужасных способов записи IP-адресов, о которых раньше не знал. Давайте изучим их вместе!

Начнём с самого простого, того, что я называю «канонической формой» IPv4 и IPv6: `192.168.0.1` и `1:2:3:4:5:6:7:8`. В разных спецификациях они называются «dotted quad» (а конкретнее «dotted decimal»), разделёнными точками полями, каждое из которых содержит 1 байт и разделёнными двоеточиями полями «colon-hex», каждое из которых содержит 2 байта.

Первые сложности возникают из-за IPv6. В канонической форме посередине многих адресов возникают длинные последовательности нулей. Поэтому обозначение `::`
позволяет пропустить один или несколько 16-битных блоков нулей: `1:2::3:4` означает `1:2:0:0:0:0:3:4`.

Во-вторых, так сложилось исторически, что IPv6 позволяет записывать последние 32 байта адреса в виде dotted quad. По сути, можно приклеивать адрес IPv4 в конец адресов IPv6!
`1:2:3:4:5:6:77.77.88.88` означает `1:2:3:4:5:6:4d4d:5858`.

И, разумеется, можно комбинировать оба подхода! `fe80::1.2.3.4` означает `fe80:0:0:0:0:0:102:304`

Существование `::` добавляет в парсинг раздражающий пограничный случай:
`::` может находиться в начале или конце адреса, а «пустая» сторона адреса может и не быть одним из 16-битных полей. `::1` означает `0:0:0:0:0:0:0:1`, `1::` означает `1:0:0:0:0:0:0:0`, а `::` означает `0:0:0:0:0:0:0:0`. Это естественным образом следует из правила `::`, однако немного усложняет написание парсера.

Последнее правило для IPv6: строго говоря, каждое поле colon-hex состоит из 4 шестнадцатеричных чисел, но можно опустить нули в начале, что я и сделал выше. В полной канонической форме `::` обозначает
`0000:0000:0000:0000:0000:0000:0000:0000`. Прошу прощения у всех читателей-трипофобов.

В целом, с IPv6 разобрались. Теперь перейдём к IPv4!

Забавный факт: текстовое представление IPv4 не было стандартизировано ни в одном документе до того, как стандарту IPv6 не потребовалась грамматика для его странной записи «trailing dotted quad». То есть это стандарт де-факто, смысл которого в основном сводится к «что понимал 4.2BSD?» и «как поступали другие ОС при копировании у 4.2BSD?»

А уж в 4.2BSD использовались довольно эксцентричные решения! Давайте для примера возьмём `192.168.140.255`. Это адрес IPv4, на который люди смотрят и понимают, что это действительно адрес IPv4. Как ещё можно записать этот же самый адрес?

Вот тот же самый IP-адрес: `3232271615`. Мы получаем его, интерпретировав 4 байта IP-адреса как беззнаковое 32-битное целое число в big-endian. Этот приводит к возможности классического дешёвого трюка: если попробовать зайти на http://3232271615, Chrome загрузит http://192.168.140.255.

Ну ладно, но ведь это как будто логично? Адрес IPv4 — это 4 байта, поэтому запись его как одного числа немного неудобна для человека, но в целом вполне возможна?

А как насчёт `0300.0250.0214.0377`? Это всё тот же адрес. Dotted quad, только теперь каждое поле записано в восьмеричной системе.

Если поддерживается восьмеричная система, то можно подумать и задаться вопросом о шестнадцатеричной. И вы будете правы! Согласно правилам 4.2BSD, `192.168.140.255` записывается и как `0xc0.0xa8.0x8c.0xff`.

Теперь вспомним о том, что когда-то у нас был CIDR (Classless Inter-Domain Routing). Адреса IPv4 имели классы: Class A, Class B или Class C. Странное было время.

И это странное время нашло отражение в IP-адресах! Знакомая нам запись `192.168.140.255`, строго говоря, является записью «Class C». Можно также записать этот адрес в формате «class B» как `192.168.36095`, или в формате «Class A» как `192.11046143`. Здесь мы объединяем последние байты адреса или в 16-битное, или в 24-битное целочисленное поле.

Кстати, из-за этого утилиты типа `ping` могут принимать странно выглядящие адреса наподобие `127.1` вместо `127.0.0.1`. В отличие от IPv6, они не реализуют расширение по типу «пропущенные поля — это нули». `127.1` — это запись Class A, обозначающая «хост 1 сети 127», где 1 — это 24-битное число.

Наконец, мы добрались до последней особенности не указанного в спецификациях поведения: можно ли в каждом из четырёх чисел адресов IPv4 использовать неограниченное количество нулей в начале? Или максимум — это три цифры? Форма `001.002.003.004` считается допустимой. А как насчёт `0000000001.0000000002.0000000003.000000004`?

Также можно задаться вопросом, должны ли какие-то из этих чисел читаться как восьмеричные, потому что ранее мы говорили, что нули в начале могут интерпретироваться как восьмеричные. Ответ: это зависит от ситуации! Существуют реализации, где используются обе формы, однако *в большинстве* современных реализаций отказались от восьмеричной и шестнадцатеричной записи, а нули в начале считаются десятичными.

Споры о нулях в начале частично заразили и IPv6. Является ли `000001::00001.00002.00003.00004` допустимым адресом IPv6 (его «стандартный» вид — `1::1.2.3.4` или `1::102:304`)? Похоже, большинство современных парсеров допускает бесконечное количество нулей в начале, вероятно, потому что используют какую-то библиотеку парсинга integer, реализующую такое поведение.

Итак, мы пришли к неутешительному выводу. Если вы хотите реализовать *правильный* парсинг IP-адресов, то вам придётся иметь дело со всей этой галиматьёй.

На данный момент мой медленный парсер обрабатывает с помощью jettison кучу старого багажа, и придерживается того, что я считаю разумным подмножеством всех этих вариантов.

- Он понимает классический IPv4 с разделёнными точками десятичными значениями и любым количеством нулей в начале.
- Он не обрабатывает запись Class A/B, а также шестнадцатеричную и восьмеричную запись.
- Он не обрабатывает запись uint32.
- Что касается IPv6, то он понимает каноническую форму colon-hex, а также `::` и стиль с записью IPv4 в конце (где IPv4 в конце следует тем же правилам, что описаны выше). В каждом поле допускается любое количество нулей в начале.

Я в сомнениях относительно последнего пункта — «формы IPv6 со встроенными десятичными числами, разделёнными точками». Парсер, который я взял за основу (`net.ParseIP` из Go) понимает его, но в реальном мире эта запись практически бесполезна. В начале развития IPv6 идея заключалась в том, что можно «проапгрейдить» адрес до IPv6, добавив в начале пару двоеточий, например, `::1.2.3.4`, однако современные механизмы перехода больше не обеспечивают столь чётко выраженного, поэтому в реальности такая запись почти не встречается.

Теги: парсинг, парсер, IP-адрес, timeweb

Хабы: Блог компании Timeweb, Ненормальное программирование, Программирование

+2 14 1,2k Комментировать Поделиться

Timeweb
VDS и инфраструктура. По коду Habr10 — 10% скидка

49,0 Карма 20,0 Рейтинг

@tw_community
Редактор блога Timeweb

ПОХОЖИЕ ПУБЛИКАЦИИ

20 января 2021 в 10:36
Таков путь! Эволюция бэкапов в Timeweb: от rsync до ZFS
22 7,8k 44 22

6 ноября 2020 в 18:42
Хост KVM в паре строчек кода. Примеры на C++ и на Python от эксперта Timeweb
39 9,2k 74 9

30 сентября 2020 в 13:26
Just add some Salt. Опыт Timeweb
5 2k 13 13

ВАКАНСИИ КОМПАНИИ TIMEWEB

Больше вакансий компании

Комментарии 0

САМОЕ ЧИТАЕМОЕ

Сутки | Неделя | Месяц

Почему практически бесполезно делать локомотив мощнее

+156 48,6k 55 243

Как Airbnb скрывает преступления при помощи тайной команды «чистильщиков»

+34 27,5k 56 96

Надували, надуваем и будем надувать. Пузыри программистов

+85 34,1k 98 59

PHP — я бы купил это за доллар

+18 13,8k 25 69

Как фронтенд-тимлиды чинили Workload

Турбо