Programmation réseau en Java : adresses IP

Michel Chilowicz

Master 2 TTT
Université Paris-Est Marne-la-Vallée

Version du 31/01/2013

Plan

Deux protocoles IP (Internet Protocol) co-existants sur le net

IPv4

Première version historique décrite dans la RFC 791 en 1981
Espace d'adressage de taille 2^{32}

IPv6

Nouvelle version introduite en 1996 pour résoudre la pénurie d'adresses IPv4
Espace d'adressage de taille 2^{128}
Version en cours d'adoption

Les adresses IPv4 et IPv6 sont attribuées par l'Internet Assigned Numbers Authority (IANA)

Les adresses IPv4

- Codage sur 32 bits, exprimée textuellement par 4 octets décimaux : $w.x.y.z$
- Un réseau est identifié par un préfixe d'adresse IP. Ce préfixe peut être exprimé :
  - Par un couple (adresse IP, masque) où masque contient des bits 1 pour les bits correspondant à l'adresse réseau
  - Par un couple (adresse IP/longueur du préfixe réseau en bits)
- Par exemple RENATER possède le bloc réseau 193.48.0.0/14 (soit un masque de 255.252.0.0), soit $2^{32-14} = 2^{18} = 262144$ adresses attribuables.

Les blocs d'adresses IPv4

5 classes d'adresses historiques publiquement adressables (à partir des 4 bits de poids fort du premier octet) avec quelques plages exceptionnelles :

- 0... : adresses de classe A avec réseau sur 1 octet (de 0.0.0.0 ... à 127.255.255.255)
  - adresse joker 0.0.0.0 : pour la configuration de socket
  - plage privée 10.0.0.0/8 (2^{24} machines adressables)
  - plage localhost 127.0.0.0/8 (machine locale)
- 10... : adresses de classe B avec réseau sur 2 octets (de 128.0.0.0 ... à 191.255.255.255)
  - plage d'adresses de lien local 169.254.0.0/16
  - plage privée 172.16.0.0/12 (2^{20} machines adressables)
- 110... : adresses de classe C avec réseau sur 3 octets (de 192.0.0.0 ... à 223.255.255.255)
  - plage privée 192.168.0.0/16 (2^{16} machines adressables)
- 1110... : adresses de classe D utilisées pour le multicast (de 224.0.0.0 ... à 239.255.255.255)
- 1111... : adresses de classe E réservées pour des usages futurs (de 240.0.0.0 ... à 255.255.255.255)
  - adresse 255.255.255.255 réservée pour le broadcast sur un réseau local

Remarque : en pratique, notions de classe obsolète (supplanté par CIDR)

Les adresses IPv6 [RFC 2373]

- Codage sur 128 bits, exprimée par groupements de deux octets en hexadécimal
- Règles de simplification :
  - Les zéros en tête de groupe sont optionnels
  - Une séquence de groupes nuls peut être mis en ellipse
  - L'adresse doit être entourée de crochets dans une URL
- Exemple d'adresse IPv6 (site web du W3C) : 2001:200:1c0:3601::53:1 $\equiv$ 2001:0200:01c0:3601:0000:0000:0053:0001 $\longrightarrow$ URL pour accéder au serveur web : http://[2001:200:1c0:3601::53:1]
- Familles d'adresses :
  - Unicast (ou anycast) : 64 bits de réseau (réseau, sous-réseau), 64 bits d'interface
    Les bits d'interface identifient une machine sur le réseau local (@MAC, DHCPv6...)
  - Multicast : 1er octet ff, second octet indiquant des propriétés sur le multicast (adresse permanente, temporaire, portée du multicast)

Quelques plages d'adresses unicast IPv6 spéciales

- Adresse nulle (::) : adresse joker utilisée pour la configuration de sockets
- ::1 : adresse localhost (boucle locale)
- fe80::10/10 : plage d'adresses de lien local
- fc00::/8 : plage d'adresses de portée privée
- ::ffff:0:0/96 : plage d'adresses transitionnelle IPv4

`java.net.InetAddress` : une adresse IP en Java

- Deux classes dérivées pour les adresses IPv4 (Inet4Address) et IPv6 (Inet6Address)
- Pas de constructeurs pour initialiser une adresse IP mais des méthodes statiques :
  - Depuis une adresse sous forme textuelle : `InetAddress.getByName(String addr)`
  - Depuis une adresse sous forme octale : `InetAddress.getByAddress(byte[] addr)`
  - Si l'on souhaite obtenir l'adresse localhost : `InetAddress.getLocalHost()`

Quelle est donc cette adresse mystérieuse ?

```
public static void d(String s) { System.out.println(s); }

public static displayAddressInfo(byte[] addr)
{
        // D'abord on récupère un objet InetAddress
        InetAddress ia = InetAddress.getByAddress(addr);
        // On affiche l'adresse sous forme textuelle
        d("Adresse texte: " + ia.getHostAddress());
        // On rappelle l'adresse octale
        d("Adresse octale: " + Arrays.toString(ia.getAddress()));
        // Ensuite on affiche les informations associées
        d("Adresse joker ? " + ia.isAnyLocalAddress());
        d("Adresse de lien local ? " + ia.isLinkLocalAddress());
        d("Adresse de boucle locale ? " + ia.isLoopbackAddress());
        d("Adresse de réseau privé ? " + ia.isSiteLocalAddress());
        d("Adresse de multicast ? " + ia.isMulticastAddress());
        if (ia.isMulticastAddress())
        {
                // Testons la portée du multicast
                d("Portée globale ? " + ia.isMCGlobal());
                d("Portée organisationnelle ? " + ia.isMCOrgLocal());
                d("Portée site ? " + ia.isMCSiteLocal());
                d("Portée lien ? " + ia.isMCLinkLocal());
                d("Portée noeud ? " + ia.isMCNodeLocal());
        }
}
```

Conversion d'une adresse IPv4 octale en texte

Méthode paresseuse

```
public String convert(byte[] addr) throws UnknownHostException
{
        return InetAddress.getByAddress(addr).getHostAddress();
}
```

Méthode plus laborieuse

```
public int toUnsigned(byte b) { return (int)b & 0xff; }
public String convert(byte[] addr) throws UnknownHostException
{
        return String.format("%d.%d.%d.%d",
                toUnsigned(addr[0]), toUnsigned(addr[1]),
                toUnsigned(addr[2]), toUnsigned(addr[3]));
}
```

Conversion d'une adresse IPv4 texte en octale

Méthode paresseuse

```
public byte[] convert(String textAddr) throws UnknownHostException
{
        return InetAddress.getByName(textAddr).getAddress();
}
```

Méthode plus laborieuse

```
public byte[] convert(String textAddr)
{
        String[] split = textAddr.split("\\.");
        byte[] addr = new byte[split.length];
        for (int i = 0; i < addr.length; i++)
                addr[i] = (byte)Integer.parseInt(split[i]);
        return addr;
}
```

Test de connectivité d'une adresse IP

La méthode `boolean isReachable(int timeout) throws IOException` d'une instance d'`InetAddress` permet de tester la connectivité de l'adresse.

Comment ça marche ?

1. On envoie soit des messages *ICMP echo request* (ping), soit on tente d'ouvrir une connexion TCP sur le port 7 (service echo) de l'hôte
2. L'hôte est considéré accessible ssi une réponse parvient avant *timeout* ms

Cas de réponses négatives

- Machine hors-ligne
- Machine ne répondant pas aux pings et sans service TCP echo
- Firewall filtrant rencontré sur le chemin de routage

Domain Name System

- Mémoriser des adresses IPv4 est difficile, des adresses IPv6 une torture
  $\longrightarrow$ nécessité d'un système d'indirection liant des noms faciles à retenir à des adresses IP
  $\longrightarrow$ solution proposée : Domain Name System (DNS) avec la RFC 1035
- Système hiérarchique de nommage chapeauté par l'Internet Corporation for Assigned Names and Numbers (ICANN) $\longrightarrow$ chaque zone DNS délègue l'admnistration de sous-noms à d'autres autorités
- Exemple de délégation : `igm.univ-mlv.fr.`
  - Gestion de `.` par l'ICANN
  - Gestion de `.fr.` par l'AFNIC
  - Gestion de `.univ-mlv.fr.` par l'Université Paris-Est Marne-la-Vallée
  - Gestion de `.igm.univ-mlv.fr.` par l'Institut Gaspard-Monge
- Conversion d'un nom en adresse IP par résolution DNS
  - On interroge les serveurs des autorités du plus général au plus spécialisé
  - Serveurs cache permettant une résolution récursive (serveur DNS du FAI)
- Résolution inverse (@IP $\rightarrow$ nom)
  - Interrogation sur le domaine `z.y.x.w.in-addr.arpa` pour une adresse IPv4 w.x.y.z
  - Interrogation sur le domaine `X.ip6.arpa` pour une adresse IPv6 (X est la succession de chiffres hexadécimaux en ordre inverse séparés par des points)

Enregistrements DNS

Couples de clé/valeur pour une ressource sur un serveur DNS :

- A : adresse IPv4
- AAAA : adresse IPv6
- CNAME : nom canonique (la ressource demandée est un alias)
- MX : serveur de courrier électronique gérant les message entrants
- SRV : serveur gérant un service spécifique du domaine
- NS : serveur de délégation d'une zone DNS
- SOA : information sur l'autorité gérant la zone DNS
- PTR : champ pointeur vers un nom canonique utilisé pour la résolution inverse
- TXT : champs de texte divers

Remarques

Certains champs peuvent exister en multiples exemplaires (plusieurs adresses IP pour équilibrage de charge, serveurs de mails pour redondance)
Il existe d'autres champs spécialisés utilisés par des extensions de DNS (DNSSEC) ou d'autres protocoles annexes (authentification de emails avec SPF, empreinte SSH avec SSHFP...).

Résolution de nom

Exécutables utiles

- `host name` : affiche les adresses IP liées à *name* en utilisant le serveur et cache DNS système
- `dig @dnsserver name` : récupère les entrées DNS de *name* auprès du serveur *dnsserver*

Appel système POSIX

`getaddrinfo()` (`netdb.h`)

Adresse IP depuis un nom

- La méthode statique `InetAddress.getByName(String nom)` fonctionne aussi bien pour une @IP textuelle qu'un nom de domaine : elle retourne **une** `InetAddress` résolue (résolution bloquante).
- `InetAddress.getAllByName(String nom)` retourne **toutes** les adresses IP correspondant au nom de domaine donné sous la forme d'un tableau d'`InetAddress` résolues.
- `InetAddress.getByAddress(String nom, byte[] addr)` retourne une `InetAddress` associant le nom et l'adresse spécifiés sans vérification de concordance.

Une exception `UnknownHostException` est levée en cas de problème (nom d'hôte non résolvable).

Méthode utiles d'`InetAddress`

- `String getHostName()` retourne le nom d'hôte (peut aussi permettre une résolution inverse)
- `String getCanonicalHostName()` permet d'obtenir le nom canonique (CNAME)

Implantation d'un résolveur DNS en Java

```
public static void main(String[] args)
{
        if (args.length < 1)
                throw new IndexOutOfBoundsException("A domain must be provided");
        String name = args[0];
        InetAddress[] addrs = InetAddress.getAllByName(name);
        for (InetAddress addr: addrs)
                System.out.println(addr.getHostAddress());
}
```