DNS再入門

2002年12月19日 発表
2003年1月16日 最終更新
Internet Week 2002/DNS DAY
(株)日本レジストリサービス(JPRS)
森下 泰宏
<yasuhiro@jprs.co.jp>

内容

- DNSの「設定」が意味するもの
  - 設定の意味を「理解したうえで」設定しているか
- DNSの運用・実装に関するいくつかの問題
  - いくつかの「実装依存」な事項についての考察

DNSの「設定」が意味するもの

- SOAの数字
  - SOA TTLと$TTL
- 外部名の設定が意味するもの
  - NS
  - MX
  - CNAME

SOAの数字

- **それぞれの数字の「意味」**
- SOA**の例**

```
$ORIGIN example.jp.
$TTL        86400
@           IN SOA  ns1.example.jp. hostmaster.example.jp. (
                    2002121901      ; serial
                    86400           ; refresh
                    21600           ; retry
                    2419200         ; expire
                    1200            ; minimum
                    )
```

SOAの数字の意味

- Serial (**重要度**:**大**)
  - **シリアル番号**
  - **ゾーンの「バージョン番号」**
  - **更新の際、手でバージョンを上げてやる必要あり**(BIND**の場合**)
    - BIND**では、**YYYYMMDDnn **が推奨されている**(RFC1912)
    - djbdns**では、ファイルのタイムスタンプから自動生成される**(**手作業不要**)
- Refresh (**重要度**:**小**)
  - **プライマリに対する定期的なゾーンチェック**(Serial)**の間隔**
  - **「昔」は**20**分～**2**時間**(**ネットワークが充分速い場合**)**とされていた**
    - **現在の**BIND**では**Notify**機構があるため、あまり気にしなくて良い**
    - djbdns**ではゾーン転送機構を使用しないため、気にしなくて良い**
  - **機構がある場合、**1**日あるいはそれ以上にしてよい、とある** (RFC1912)

SOAの数字の意味

- Retry (重要度:小)
  - Refreshによって設定されたインターバルでプライマリに接続できなかった時に、再試行(Retry)する間隔
  - Refreshと同様、現在ではあまり気にしなくて良い
  - Refreshの数分の1に設定しておけば問題ない
- Expire (重要度:小)
  - ゾーン転送に失敗し続けた際に、そのゾーンを無効とみなすまでの期間
  - セカンダリサーバのサービスをしているプロバイダでは、効いて来る場面もある
  - 通常は2〜4週間(RFC1912)に設定しておけば問題ない
  - Expireの値はRefreshより必ず大きくしておくこと

SOAの数字の意味

- Minimum (**重要度**:**大**)
  - **最小**TTL**パラメータ**
  - **「昔**(RFC1035)**」と「現在**(RFC2308)**」では「意味が違う」**
    - **しかし、このことが広く認識されているようには見えない**
    - **特に、昔から設定しているところに誤解(というより、無意識による不適切な設定)が広く出回っている**

SOA TTLと$TTL

- SOA TTL(SOAレコードで設定するMinimum)の意味
  - 昔(RFC1035,RFC1918)
    - そのゾーンのRRのデフォルトのTTL
  - 現在(RFC2308)
    - ネガティブキャッシュのTTL
    - そのゾーンファイルのデフォルトTTLは$TTLで別途設定
      - BINDの場合

SOA TTLと$TTL

- 昔、SOA TTLは1〜5日が適切とされてきた(RFC1912)
  - 今でもこの設定をしているドメインが多く見られる
    - 特に「1日(=86400)」にしているところが多い
- 現在ではSOA TTLはネガティブキャッシュのTTLとして解釈される(RFC2308)
  - ネガティブキャッシュ: その名前がなかった、ということのキャッシュ
  - この値が大きい場合、新規に登録した名前が最悪の場合SOA TTLで設定した時間牽けなくなる
    - つまり、1日に設定している場合、最悪1日名前が牽けなくなる
    - 特に登録したい人が「設定されたかな」と思ってチェックした場合、その時点でネガティブキャッシュに入ることになる
    - ネームサーバを再起動すると「牽けるようになる」ので、適当に再起動してお茶を濁してしまう(原因を究明しない)場合が多い
    - (2003年1月16日更新)最近のBINDやdjbdnsのキャッシュサーバでは、不適切に大きな値のネガティブキャッシュTTLは無効になる

TTLをどう設定すべきか

- BINDの場合
  - SOA TTL
    - 数10分程度(RFC2308のサンプルでは20分(=1200))
  - $TTL
    - 1日〜3日(RFC2308のサンプルでは1日(=86400))
- djbdnsの場合
  - 自動的に適切な値が設定される(指定することも可能)
    - SOA TTL: 2560 (42分40秒)
    - 通常のレコードのデフォルトTTL: 86400(1日)
    - NSとNSで指定されるAのデフォルトTTL: 259200(3日)

まとめ(SOAの数字)

- **重要なのはバージョン**とMinimum TTL
- **ネガティブキャッシュの意味と動作を理解しておくこと**(RFC2308**を読みましょう**)
- TTL**の適切な設定値**
  - SOA TTL: **数**10**分**
    - **ただし、サーバの性格により一概に言えない**
    - **数時間程度にした方がいい場合もある**
  - **ゾーンのデフォルト**TTL($TTL): 1**〜**3**日**

外部名の設定が意味するもの

- 「外部名」とは何か
  - 自分のゾーン内の名前以外の指定
- 考慮すべきRR
  - NS, MX, CNAME

自分のゾーン以外のNS指定

- **プロバイダにセカンダリサーバを頼んでいる場合等によく見られる**
- **技術的には間違いではない、しかし、、、**

- **設定例**(DNS**サーバの**1**つが外部名**)

```
$ORIGIN example.co.jp.
@           IN      NS      ns1.example.co.jp.
            IN      NS      secondary.provider.ne.jp.
```

- **設定例**2(**すべての**DNS**サーバが外部名**)

```
$ORIGIN example.co.jp.
@           IN      NS      hosting1.provider.ne.jp.
            IN      NS      hosting2.provider.ne.jp.
```

この設定の意味

- 「私は、このプロバイダのゾーン全体(provider.ne.jp)を100%信用します」ということを意味する
- もしこのプロバイダのゾーンを管理するDNSサーバがクラックされ、provider.ne.jpのゾーンを乗っ取られた場合、例えホストsecondary.provider.ne.jp自体が無事であったとしても、example.co.jpの名前空間をすべて乗っ取られるリスクがある
  - 例1のような場合でも、何らかの方法(例えば、パケットをあふれさせる)でns1.provider.co.jpを麻痺させればよい
  - 例2ではその必要もない(のでもっと簡単)

自分のゾーン以外のMX指定

- **設定例**

  $ORIGIN example.co.jp.

  @ **IN MX 10 mx1.provider.ne.jp.**

この設定の意味

- 「私はメールに関して、このプロバイダが管理するゾーン全体(provider.ne.jp)を100%信用します」ということを意味する
- もしこのプロバイダのゾーンを管理するDNSサーバがクラックされ、provider.ne.jpのゾーンを乗っ取られた場合、例えホストmx1.provider.ne.jp自体が無事であったとしても、example.co.jp宛てのメールはすべて奪われるリスクがある

自分のゾーン以外のCNAME設定

- **設定例**

  $ORIGIN example.co.jp.

  www　　**IN CNAME www1.provider.ne.jp.**

この設定の意味

- 「私はその名前(www.example.co.jp)に関して、CNAMEで指定した相手先がどのように設定を変更しても、その結果を受け入れます」ということを意味する
- このプロバイダのDNSがクラックされれば…これまでと同様なので省略
- クラックされなくても、プロバイダ側の設定ミスや事故(DNSサーバのダウン)等による影響を直接受けることになる

キャッシュサーバから見た「外部名」

- 名前解決にかかるコスト(負荷、時間)が増大する
  - いったんリクエストを「棚上げ」して、NS(あるいはMX)で指定されたホストに関する名前のNSとAを調べなければならないため
- 甚だしい場合、名前解決自体にも影響
  - 「グルーなし」(外部NS参照)が多段に渡ると、名前解決自体が失敗することもある
    - 3段を超えるとかなり危ない

どう設定すればよいのか(NS)

- **同じIPアドレスを持った「内部名」でNS用の名前(A)を指定し、それをNSで指定する**
- **例**(NS):

```
$ORIGIN example.co.jp.
@               IN      NS      ns1.example.co.jp.
                IN      NS      secondary.example.co.jp.
…
secondary       IN      A       xxx.xxx.xxx.xxx ;DNSサーバのIPアドレス
```

  - **実際には**secondary.example.co.jp**はプロバイダ内にある**secondary.provider.ne.jp**と同一ホストを指定**
  - **レジストリには、この名前(内部名)で登録**
  - **この場合、**secondary**の**A**を**example.co.jp**ゾーン内で記述**

どう設定すればよいのか(MX)

- NSと同様、「内部名」でMX用の名前(A)を指定し、それをMXで指定する
- 例(MX)

```
$ORIGIN example.co.jp.
@       IN      MX 10   mx1.example.co.jp.
mx1     IN      A       xxx.xxx.xxx.xxx ;メールサーバのIPアドレス
```

  - mx1.example.co.jpはプロバイダ内にあるmx1.provider.ne.jpと同一ホストを指定
  - mx1のAをexample.co.jpゾーン内で記述

どう設定すればよいのか(CNAME)

- 無用なCNAMEを使うのをできるだけ避ける
  - Aで指定すればCNAMEは事実上必要ない
  - 運用上の理由で指定する場合でも、外部名をCNAMEで参照することはやめる
    - 外部のDNSサーバの影響を直接受ける
    - 外部から見た場合「CNAMEで指定されている」ことを管理しなければならなくなる(面倒)

まとめ(外部名)

- NS, MXには内部名を設定
  - 名前空間やメールを奪われる(無用な)リスクを避けられる
  - このような設定をさせてもらえる(してくれる)プロバイダは「DNSに気を使っている、サービス品質のよいプロバイダ」であると(少なくとも私個人は)みなします
- CNAMEはできるだけ使用しない
  - 使用する場合も、同一ゾーン内の参照にとどめる

DNSの運用・実装に関するいくつかの問題

- クラスレスin-addr.arpaドメインの委譲
- NSレコードの取り扱い
  - NSレコードはどのようにキャッシュされるべきか
- DNSキャッシュの安全性
  - DNS query ID, ポート番号
- MS DNSの問題

クラスレスin-addr.arpaドメインの委譲

- RFC2317(BCP)で規定
  - CNAMEを使ったやり方
- 技術的には間違いではない、しかし、、、
- さまざまなトラブルの原因になっている
  - プロバイダによって設定内容が微妙に違う
    - 委譲の仕方
    - ネット部(一番最初)とブロードキャスト(一番最後)、あるいはルータのIPアドレスを委譲するかしないかが、プロバイダによって違う
  - ユーザが設定の原理を理解していない
    - そもそも、しくみを理解しにくい

おさらい: CNAMEを使ったやり方 (BINDの場合)

- 例:192.168.123.136/29
- 上位プロバイダ(123.168.192.in-addr.arpa)では、以下のように設定

```
$ORIGIN     123.168.192.in-addr.arpa.
@           IN      SOA     …
            IN      NS      …
subnet136   IN      NS      ns1.example.co.jp.
136         IN      CNAME   136.subnet136.123.168.192.in-addr.arpa.
137         IN      CNAME   137.subnet136.123.168.192.in-addr.arpa.
138         IN      CNAME   138.subnet136.123.168.192.in-addr.arpa.
139         IN      CNAME   139.subnet136.123.168.192.in-addr.arpa.
140         IN      CNAME   140.subnet136.123.168.192.in-addr.arpa.
141         IN      CNAME   141.subnet136.123.168.192.in-addr.arpa.
142         IN      CNAME   142.subnet136.123.168.192.in-addr.arpa.
143         IN      CNAME   143.subnet136.123.168.192.in-addr.arpa.
```

おさらい: CNAMEを使ったやり方 (BINDの場合)

- **顧客側では、以下のように設定**

```
$ORIGIN     subnet136.123.168.192.in-addr.arpa.
@    IN     SOA    …
     IN     NS     ns1.example.co.jp.
136  IN     PTR    cust-net.example.co.jp.
137  IN     PTR    router.example.co.jp.
138  IN     PTR    host1.example.co.jp.
139  IN     PTR    host2.example.co.jp.
140  IN     PTR    host3.example.co.jp.
141  IN     PTR    host4.example.co.jp.
142  IN     PTR    host5.example.co.jp.
143  IN     PTR    broadcast.example.co.jp.
```

おさらい: CNAMEを使わないやり方 (BINDの場合)

- 例:192.168.123.136/29
- 上位プロバイダ(123.168.192.in-addr.arpa)では、以下のように設定

```
$ORIGIN     123.168.192.in-addr.arpa.
136  IN     NS     ns1.example.co.jp.
137  IN     NS     ns1.example.co.jp.
138  IN     NS     ns1.example.co.jp.
139  IN     NS     ns1.example.co.jp.
140  IN     NS     ns1.example.co.jp.
141  IN     NS     ns1.example.co.jp.
142  IN     NS     ns1.example.co.jp.
143  IN     NS     ns1.example.co.jp.
```

おさらい: CNAMEを使わないやり方 (BINDの場合)

- **顧客側では、以下のように設定**

```
$ORIGIN    136.123.168.192.in-addr.arpa.
@    IN    SOA    …
     IN    NS     ns1.example.co.jp.
     IN    PTR    cust-net.example.co.jp.

$ORIGIN    137.123.168.192.in-addr.arpa.
@    IN    SOA    …
     IN    NS     ns1.example.co.jp.
     IN    PTR    router.example.co.jp.
```

…**以下、ホスト数分だけ繰り返し**

named.conf**も同様に設定必要**

クラスレスin-addr.arpaドメインの委譲

- なぜCNAMEを使う設定が「標準」になったのか

⇒多分にBINDに依存

  - BINDでCNAMEを使わない場合、ちょっと面倒な設定をしなければいけない
- このRFCが発行された1998年当時は、事実上BINDしか使える実装がなかった(ので、Best ”Current” Practiceとしてリリースされた…)
  - BINDでは、
    - CNAMEを使わない場合、顧客側はIPアドレスの数だけ、
      - named.confでゾーンを宣言し
      - 逆引き1つだけのゾーンを一つ一つ定義する

      必要がある
    - CNAMEを使う方法では、顧客側は通常の逆引きに近い形で、named.confやゾーンファイルを記述できる

現実はこうである、しかし、、、

- BINDを使う場合でも、CNAMEを使わないやり方の方がしくみを理解しやすい(と思う)
  - 通常のゾーン委譲と同様に取り扱える
    - @にPTRを書くということがとっつきにくいかもしれないが、原理を理解していれば、決して特殊なことではない
  - CNAMEを使うことによる無用なトラブルも起こりにくい
- プロバイダのやり方による「違い」が出ない
  - 設定の間違いが起こるリスクが少なくて済む
  - 一般的な方法として浸透していれば、設定の面倒さはそれほど問題にならなかったのではないか

まとめ

- できれば、CNAMEを使わずに単に委譲する(まともな)やり方もサポートしてほしい(と希望します)
  - その方が間違いが少なくてすむはず

NSレコードの取り扱い

- NSレコードはどのようにキャッシュされるべきか
- 例: 上位サーバとそのサーバでTTLが違う場合
  - 上位サーバ(jp)

```
$ORIGIN jp.
$TTL    86400
example IN      NS      ns1.example.jp.
```

  - そのサーバ(example.jp)

```
$ORIGIN example.jp.
$TTL    259200
@       IN      NS      ns1.example.jp.
```

- この場合、DNSキャッシュがexample.jpのNSを要求された場合どのレコード(TTL値)がキャッシュされるべきか
  - 上位サーバの86400?
  - そのサーバの259200?

実際の動作

- BIND 8.3.4
  - **上位サーバの**86400**でキャッシュされ、そのサーバから**NS**を受信しても差し替わらない**
- BIND 9.2.1
  - **最初の**NS query**では上位サーバの**86400**でキャッシュされ、**MX**や**A**の**query**の際にそのサーバから**NS**を受信した時点で、**259200**に差し替わる**
- BIND 9.3.0-20021115
  - **最初の**NS query**の時点で「そのサーバ」にアクセスし、**259200**でキャッシュされる**
- djbdns
  - **(2003年1月16日訂正)最初の**NS query**では上位サーバの**86400**でキャッシュされ、**MX**や**A**の**query**の際にそのサーバから**NS**を受信した時点で、**259200**に差し替わる**(BIND9.2.1**と同じ**)

NSレコードの取り扱い

- RFC1034にはこう書かれている
  - The RRs that describe cuts around the bottom of the zone are NS RRs that name the servers for the subzones. Since the cuts are between nodes, these RRs are NOT part of the authoritative data of the zone, and should be exactly the same as the corresponding RRs in the top node of the subzone.
- つまり、「そのゾーン」のデータをキャッシュしなければならないように読める
  - BIND 9.3.0-20021115, djbdnsの動作が正しいように読める
- BIND 8.xの実装は、場合によっては事故につながるかもしれない(検証が必要)

DNSキャッシュの安全性

- 2002年11月にCERT/CCから出たKnowledge Base (KB):
  - Vulnerability Note VU#457875: Various DNS service implementations generate multiple simultaneous queries for the same resource record
    - http://www.kb.cert.org/vuls/id/457875
  - これによるとBINDのDNSキャッシュサーバはこれに対して「Vulnerable」とある
  - そして、2002年12月5日現在も「Vulnerable」のまま

どう安全ではないのか

- BIND (4, 8, 9すべて)のDNSキャッシュサーバの実装では、問い合わせの際に常に同じUDPポート番号を使っている
- このため、外部からのbrute-force attackに対するリスクが大きい
  - 昔のBINDではDNS query IDもランダムではなかったため、さらにリスクが大きかった(これは修正された)
  - ISCでは「DNSSECの普及」が根本的解決だと言っているが、、、

リスクを軽減させる方法

- query毎にポート番号をランダムに変化させることで、このリスクを大幅に軽減させることができる(が、BINDではこのように実装されていない)
  - DNS query ID(16ビット)のみ: 2^{16}
  - ポート番号も変化: 2^{32}
  - ちなみにdjbdnsでは、このように実装されている
- そもそもDNSのquery IDが16ビットでなければ、このリスクは発生しにくかったので、プロトコル上の欠陥とも言える
  - しかし、できる対応ならそのようにすべき

MS DNSの問題

- Microsoft DNSのキャッシュサーバは、デフォルトで「毒」を受け取ってしまう
  - cache poisoningに対する脆弱性レベルが、1997年より前のBINDと同じ
- レジストリを設定することで、この設定をある程度回避することができる
  - が、デフォルトではそうなっていないので注意が必要
  - DNS キャッシュ破壊の防止策
    - http://support.microsoft.com/default.aspx?scid=kb;ja;jp241352

MS DNSの問題 (2002年12月18日追加)

- **事態はもっと深刻らしい**
  - **送信先のIPアドレスだけではなく、どのIPアドレスからのDNS返信パケットであっても、(デフォルトでは)受けてしまう**
    - **レジストリの設定で回避可能とのこと**(MSのWeb Page)
    - Microsoft Windows domain name resolver service accepts responses from non-queried DNS servers by default
      - http://www.kb.cert.org/vuls/id/458659
  - DNS Query IDが**マッチしなくても、しつこく送ると(偽の)受信パケットを受け取ってしまうらしい**
    - **実験で確認した人がいる**
  - **参照**URL
    - http://archives.jwntug.or.jp/public/index.html?ng=jwntug%2Epublic%2Esecurity&t=%3Cmid%2D25%2Dsecurity%40jwntug%2Eor%2Ejp%3E

まとめ:原典(原点)に帰ろう

- 必ずしも実装が「原典」ではない
  - 不幸にも、世の中に広く出ている実装がすべて「いい実装」とは限りません
- 特にBIND4とBIND8は、デフォルト設定や実装方法が必ずしも適切ではない
  - これはISCも認めている
- 疑問に思ったら、RFCを当たりましょう
  - でも、DNS関連のRFCでは「盲信」は禁物です
  - 「なぜ、そう書かれているのか」の心を理解しましょう
- このコマで参照したRFC(RFC2317を除く)
  - RFC1034: Domain names - concepts and facilities.
  - RFC1035: Domain names - implementation and specification.
  - RFC1912: Common DNS Operational and Configuration Errors.
  - RFC2181: Clarifications to the DNS Specification.
  - RFC2308: Negative Caching of DNS Queries (DNS NCACHE).