

IT 資格 道 場

LPIC-2 202試験 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

LPI公式 LPIC-2 Exam 202 Objectives Version 4.5 準拠

2026年10月版

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

目次

この教材群の使い方(3点セット)

第1章 DNS サーバー [207.1/207.2/207.3]

- 1-1 権威サーバーとキャッシュサーバー [207.1]
- 1-2 DNS サーバーソフトウェアの種類 [207.1]
- 1-3 named.conf:サーバー全体の設定 [207.1]
- 1-4 rndc:稼働中の named を制御する [207.1]
- 1-5 設定を壊さないための検証コマンド [207.1/207.2]
- 1-6 ゾーンファイルの構造 [207.2]
- 1-7 逆引きゾーン [207.2]
- 1-8 委任とグルーレコード [207.2]
- 1-9 マスターとセカンダリ:ゾーン転送のしくみ [207.2]
- 1-10 問い合わせの道具:dig・host・nslookup [207.2]
- 1-11 アクセス制御と情報の隠蔽 [207.3]
- 1-12 TSIG:通信相手を共有鍵で認証する [207.3]
- 1-13 DNSSEC:応答の正当性を検証できるようにする [207.3]
- 1-14 DANE と、問い合わせ自体を守る仕組み [207.3]
- 1-15 章末のまとめ [207.1/207.2/207.3]

サンプルはここまでです

この教材群の使い方(3点セット)

種類	役割
① 学習用テキスト	これだけで問題が解けるようになる本編
② 頻出度別ノート	テキストを読んだら次はこれ。頻出順に絞った問題と解説で「解ける」に橋渡し。試験直前の最終チェックにも
③ 暗記用ノート	覚えるべき要点だけを一問一答に凝縮。空き時間の反復と用語の再確認用

使い方: ① 学習用を読む → ② 頻出度別で解き方を掴む → ③ 問題演習で腕試し → ④ 頻出度別に戻って再確認(試験直前もここ)。暗記用は空き時間や用語の再確認に。

このテキストの位置づけ: ①の学習用テキストです。まずはここを通読し、これだけで問題が解けるようになることを目標にしてください。

- 対象試験: LPIC-2 202試験 (202-450、出題範囲 Version 4.5.0)。60問・90分、択一と記述入力、合格は 200～800 のスケールで 500点
- 構成: 全10章。各章の冒頭に副目標と重みを書きました(重み1につき約1問)
- 各章の末尾の「章末のまとめ」は、試験直前の見直し用です

第1章 DNS サーバー [207.1/207.2/207.3]

この章は公式副目標 **207.1 基本的な DNS サーバーの設定 (weight 3)・207.2 DNS ゾーンの作成と管理 (weight 3)・207.3 DNS サーバーのセキュリティ確保 (weight 2)** に当たります。

1-1 権威サーバーとキャッシュサーバー [207.1]

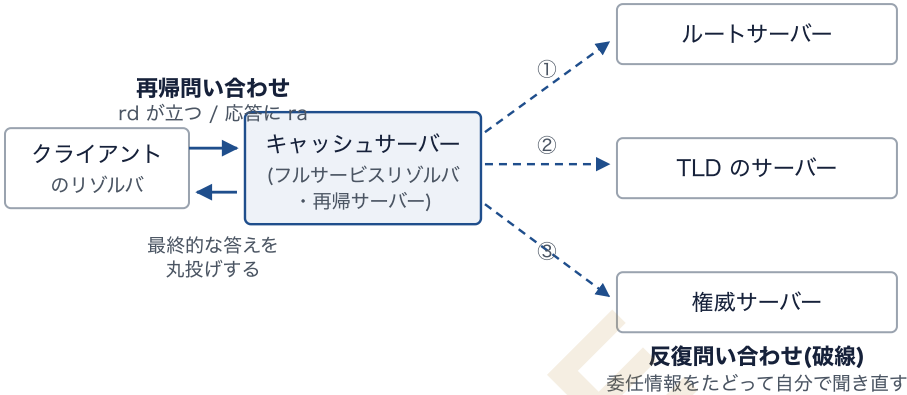
種類	役割	allow-query
権威 (authoritative) サーバー	任されたゾーンの台帳 (ゾーンファイル) を持ち、正式な答えを返す	広く開ける (公開が仕事)
キャッシュ (フルサービスリゾルバ・再帰) サーバー	台帳を持たず、権威サーバーをたどって答えを探し、一定時間ためて使い回す	利用者に絞る (踏み台にされないため)

1-1-1 再帰問い合わせと反復問い合わせ

- **再帰問い合わせ**: 「最終的な答えを持ってきて」と丸投げする形です。クライアントのリゾルバがキャッシュサーバーへ投げます (権威サーバーへは投げない)
- **反復問い合わせ**: 委任情報をたどり、自分で次のサーバーへ聞き直す形です。キャッシュサーバーがルート → TLD → 権威とたどります

問い合わせ側が再帰を要求すると **rd (Recursion Desired)** が立ちます。応答側が再帰に対応していれば **ra (Recursion Available)** が立ちます。

再帰問い合わせと反復問い合わせ —— 丸投げする側と、たどる側



クライアントとキャッシュサーバーの間だけが再帰問い合わせで、そこから先は反復問い合わせ。
権威サーバーが自分のゾーンデータから直接答えた応答にだけ aa フラグが立ち、キャッシュサーバーが返す答えには aa は立たない。

図 1-1 再帰問い合わせと反復問い合わせ:再帰はクライアントとキャッシュサーバーの間だけ

1-1-2 応答のフラグを読む

フラグ	立っている時の意味
qr	問い合わせではなく応答
aa	権威ある回答(自分のゾーンデータから直接答えた)
tc	応答が切り詰められた。リゾルバは TCP で再送する
rd / ra	再帰を要求した／再帰に対応している

;; flags: qr rd ra; のように aa が無ければ、非権威の回答です。UDP で収まらず tc が立つと、同じ問い合わせを **TCP の 53 番**へ送り直します。ゾーン転送も TCP を使います。

1-2 DNS サーバーソフトウェアの種類 [207.1]

ソフトウェア	位置づけ
BIND 9	最も広く使われる。権威とキャッシュのどちらにもなる
dnsmasq	小規模向けの軽量実装。 DNS キャッシュと DHCP サーバーを兼ねる (大規模な権威運用向けではない)
Unbound	キャッシュ(再帰)専用 。DNSSEC 検証を備える
NSD	権威専用 。再帰機能を持たない
PowerDNS	データベース (MySQL・PostgreSQL など) をバックエンドにできる

Unbound と NSD は、担当が逆です (同じ NLnet Labs 製なので取り違えを誘う)。

1-2-1 dnsmasq の設定を読む: `server=` • `address=` • `no-resolv`

dnsmasq は、小規模向けの軽量の **DNS フォワーダ(キャッシュ・転送)と DHCP サーバー**です。BIND のようにゾーンを持つ権威サーバーではありません。設定は `/etc/dnsmasq.conf` などに 1 行ずつ書きます。`dnsmasq --test` で構文を検査します (正しければ `syntax check OK`、誤りは `bad option at line N` で終了コードが 1)。

```
port=5353
listen-address=127.0.0.1
bind-interfaces
no-resolv
server=/example.jp/127.0.0.1#53
address=/test.local/192.0.2.99
host-record=nas.local,192.0.2.50
```

指定	働き
port=	待ち受けるポート(既定は 53)
server=/ドメイン/アドレス#ポート	そのドメインの問い合わせだけを指定のサーバーへ転送する
no-resolv	/etc/resolv.conf を転送先に使わない
address=/名前/アドレス	その名前(と配下)に、固定のアドレスを返す
host-record=名前,アドレス	その名前に、固定のアドレスを返す

この設定で 127.0.0.1 の 5353 番へ問い合わせると、www.example.jp は server= に合致します。そのため BIND (53 番) へ転送され、192.0.2.80 が返ります。test.local は 192.0.2.99、nas.local は 192.0.2.50 が返ります。それぞれ各行の指定どおりで、同じ値にはなりません。other.org は、server= に合致しません。no-resolv で /etc/resolv.conf も使わないので、転送先が無く **REFUSED** になります。

LPIC-2 では、dnsmasq・djbdns・PowerDNS が BIND の代わりになるネームサーバーだと知っておけば十分です。設定の細部までは求められません。

BIND 以外のソフトは、権威とキャッシュを別のプログラムに分けるものがあります。

ソフト	権威サーバー	再帰・キャッシュ
BIND	<code>named</code> (1 つで両方を兼ねる)	<code>named</code>
djbdns	<code>tinydns</code> : 手元に設定した情報で、反復問い合わせに答える	<code>dnscache</code> : ゾーンを持たない。再帰問い合わせを受けて他のサーバーに尋ね、答えをキャッシュする
PowerDNS	<code>pdns_server</code>	<code>pdns_recurser</code>
dnsmasq	(ゾーンを持つ用途には向かない)	キャッシュと転送が主な役目

`dnscache` と `pdns_recurser` は、ゾーンのマスターにはなりません。`dnsmasq` は権威専用ではなく、再帰の中継とキャッシュを担います。

1-3 `named.conf` : サーバー全体の設定 [207.1]

ステートメント	何を書くか
<code>options</code>	<code>named</code> 全体の既定 (ディレクトリ・再帰・待ち受け・アクセス制御など)
<code>zone</code>	ゾーンごとの役割・ファイル名・転送許可先
<code>acl</code>	アドレスの集合に名前を付ける
<code>logging</code>	ログの出力先と分類
<code>controls</code>	<code>rndc</code> からの制御を受け付けるアドレスと鍵

ステートメント	何を書くか
<code>view</code>	問い合わせ元で見るゾーンを切り替える (split DNS)

`options` は、全体の既定を1か所にまとめるブロックです。`zone` の内側に書くものでも、ゾーンごとに用意するものでもありません (`global` というブロックは存在しない)。

named.conf の構造 — ステートメントの集まり

`options { ... };`
named 全体に関わる既定の設定
directory / listen-on / recursion
allow-query / allow-transfer / forwarders

`zone "example.jp" { ... };`
ゾーンごとの役割・ファイル名・転送許可先

`zone "2.0.192.in-addr.arpa" { ... };`
ゾーンは複数書ける

`include "<別のファイル>;"` 別ファイルを読み込んで取り込む

`acl` アドレスの集合に名前を付ける

`key` TSIG の鍵の名前・アルゴリズム

`server` 相手サーバーごとの設定

`logging` 出力先と分類

`controls` rndc を受け付ける相手

`view` 問い合わせ元で見せ分ける

どのステートメントも { } のブロックを持ち、末尾はセミコロンで閉じる。options は全体の既定を 1 か所にまとめるブロックで、zone の内側に書くものでも、ゾーンごとに用意するものでもない。global という名前のブロックは named.conf に存在しない。

図 1-2 named.conf の構造: options は全体の既定を 1 か所にまとめる

1-3-1 options の主なディレクティブ

行	働き
<code>directory</code>	<code>zone</code> の <code>file</code> に書いた相対パスの基準
<code>allow-query</code>	通常の名前解決の問い合わせを受け付ける相手

行	働き
<code>allow-transfer</code>	ゾーン転送 (AXFR/IXFR) を許可する相手
<code>recursion</code>	再帰を受け付けるか (権威専用なら <code>no</code>)
<code>forwarders</code> / <code>forward</code>	再帰を転送する先 / 転送先が答えない時の動き
<code>version</code>	<code>version.bind</code> に返す文字列 (実際の版を隠す)
<code>notify</code>	更新時にセカンダリへ NOTIFY を送るか (既定 <code>yes</code>)
<code>dnssec-validation</code>	応答を DNSSEC で検証するか

`allow-query` と `allow-transfer` を入れ替える肢が定番です。ゾーン転送を誰にでも許すと、ゾーンの全内容が第三者に渡ります。`allow-transfer` は `options` にも `zone` にも書けます (`zone` 側が優先)。

1-3-2 `forward first` と `forward only`

値	転送先から応答が得られなかった時
<code>forward first</code> (既定)	自分でルートからの反復問い合わせに切り替える
<code>forward only</code>	自分では解決せず、 名前解決はそのまま失敗する

違いは試行回数ではなく、フォールバックするかどうかです。外部への直接の問い合わせを許さないネットワークでは、`forward only` を選びます。

1-3-3 zone 文と type

```
zone "example.jp" IN {
    type master;
    file "example.jp.zone";
    allow-transfer { 192.0.2.54; };
    also-notify { 192.0.2.54; };
};

zone "example.jp" IN { type slave; file "slaves/example.jp.zone";
masters { 192.0.2.53; }; };

zone "2.0.192.in-addr.arpa" IN { type master; file "192.0.2.rev"; };
zone "." IN { type hint; file "named.ca"; };
```

type	役割
master	プライマリ。ゾーンファイルを自分で管理する
slave	セカンダリ。masters の相手から転送で複製を受け取る
hint	ルートサーバーの一覧（ゾーン名は . ）
forward	そのゾーンへの問い合わせを別サーバーへ転送する

新しい BIND では、primary / secondary とも書けます。type tsig という値はありません。file はファイルの場所を示すだけです（保存形式は伝えない）。

1-3-4 masterfile-format :テキスト形式と生形式

masterfile-format は、ゾーンファイルの保存形式を named に伝えます。値は text (人が読む形。マスターの既定)と raw (高速に読める生形式。セカンダリが保存する既定)です。

```
zone "example.com" IN { type master; file "example.com.db"; masterfile-format raw; };
```

生形式は BIND の版に依存するので、別マシンへそのまま持ち込みません。masterfile-style は保存形式ではなく、書き出す時の名前の並べ方を選ぶ指定です。

1-3-5 ディストリビューションによる置き場所の違い

	Red Hat 系	Debian 系
主設定ファイル	/etc/named.conf に一本化	/etc/bind/named.conf から include で分割
options の置き場	/etc/named.conf の中	/etc/bind/named.conf. options
ゾーン定義の置き場	/etc/named.conf の中	/etc/bind/named.conf. local
既定の directory	/var/named	/var/cache/bind
パッケージ	bind・bind-utils	bind9・dnsutils (bind9-dnsutils)
サービス名	named	bind9 (named も別名で使える)

1-3-6 logging : ログの出し先と分類

出し先を決めるのが `channel`、どの分類をどこへ送るかを決めるのが `category` です。

```
logging {
    channel query_log {
        file "/var/log/named/query.log" versions 3 size 20m;
        severity info;
        print-time yes;
    };
    category queries { query_log; };
    category lame-servers { null; };
};
```

`severity` は、受け取る重大度の下限です。組み込みの `channel` は4つあります (`default_syslog` ・ `default_debug` ・ `default_stderr` ・ `null`)。よく出る分類は、`queries` (クエリログ)、`security` (許可と拒否)、`xfer-in` / `xfer-out` (転送)、`lame-servers` です。ある分類を捨てるなら、その `category` を `null` へ向けます。クエリログは `rndc querylog` でも切り替えられます。

1-3-7 問い合わせの記録を始める: `querylog` と書き込み権限

`versions 3 size 5m` は、ファイルが **5MB** に達したら新しいファイルに切り替え、古いものを **3 世代**だけ残す指定です。`print-time yes` が各行に付けるのは**時刻**で、問い合わせ元の IP アドレスではありません。

動いている `named` に `category` を書き足して `rndc reconfig` で読み直しても、**問い合わせの記録は始まりません**。`category` はメッセージの出力先を割

り当てただけで、記録そのものを切り替える設定ではないからです。ファイルは作られても、中身は空のままだ。`rndc reconfig` のあとに `dig` で名前を引いてもファイルは 0 バイトのままです。`rndc querylog on` を実行して初めて記録が始まります。記録を始める方法は次の 2 つです。

方法	内容
<code>rndc querylog on</code>	実行中の <code>named</code> で記録を始める(止めるのは <code>rndc querylog off</code>)
<code>options</code> の <code>querylog yes;</code>	設定ファイルで記録を有効にする

出力先のディレクトリには、`named` を動かすユーザー(Debian 系は `bind`)の **書き込み権限** が要る。`named` は `root` の権限でログを書くわけではないため、権限がなければログは出力できません。

1-3-8 named.conf の文法: 閉じ括弧のあとのセミコロン

`named.conf` 系のファイルでは、ステートメントを閉じる `}` のあとに必ず `;` を置きます。

```
1 zone "example.jp" {
2     type primary;
3     file "/etc/bind/zones/db.example.jp";
4 }
5 zone "2.0.192.in-addr.arpa" {
```

```
$ named-checkconf /etc/bind/named.conf.local
/etc/bind/named.conf.local:5: missing ';' before 'zone'
```

4 行目の `}` に `;` が無いので、`named-checkconf` は 5 行目の `zone` という語に出会った時点で「直前の文が閉じていない」と判断する。**報告される行番号は、原因の次の文が始まる位置**です。`zone` 名の引用符や、`type primary` (`BIND 9.18` でも使える現行の書き方)が原因ではありません。`named-checkconf` は既定ではゾーンファイルを読まないの、ファイルの有無は検査しません(ゾーンファイルの検査は `named-checkzone`)。

1-3-9 `listen-on` と `allow-query` :他のホストに応答させる最低限の変更

Rocky Linux などに `BIND` を入れた直後の `options` は、自分自身だけが使える状態です。

```
options {  
    listen-on port 53 { 127.0.0.1; };  
    listen-on-v6 port 53 { ::1; };  
    directory      "/var/named";  
    allow-query     { localhost; };  
    recursion yes;  
};
```

指定	決めるもの
<code>listen-on port 53 { アドレス; };</code>	どの IPv4 アドレスで待ち受けるか。IPv6 は <code>listen-on-v6</code>
<code>allow-query { ... };</code>	どのクライアントからの問い合わせに答えるか
<code>recursion</code>	再帰問い合わせに応じるか

指定	決めるもの
<code>allow-transfer</code>	ゾーン転送を許す相手 (通常の問い合わせとは別)

社内の `192.168.10.0/24` の他のホストから、このサーバー (`192.168.10.53`) を使えるようにするには、最低限、2 つの変更が要ります。1 つ目は、**待ち受けるアドレスに `192.168.10.53` を足す (listen-on)** ことです。2 つ目は、**問い合わせを許す相手に `192.168.10.0/24` を足す (allow-query)** ことです。どちらか一方だけでは応答しません。`recursion` を `no` にしても、`allow-transfer` を足しても、他のホストに答えられるようにはなりません。`directory` は `zone` の `file` に書いた相対パスの基準で、問い合わせへの応答とは関係がありません。

1-4 `rndc` :稼働中の `named` を制御する [207.1]

`rndc` は、稼働中の `named` へ制御コマンドを送ります (`stop`・`halt` 以外はプロセスを止めない)。

1-4-1 認証のしくみ

`rndc` と `named` は、**共有鍵による HMAC 認証**を行います。`rndc.conf` と `named.conf` の両方に同じ `key` を定義します (多くは `rndc.key` を両方から `include`)。

```
key "rndc-key" { algorithm hmac-sha256; secret
"aG9nZWZhV2ZVmdWdhZnVnYQ==" };

controls { inet 127.0.0.1 port 953 allow { 127.0.0.1; } keys { "rndc-key"; }; };
```

制御チャンネルは **TCP の 953 番**です。ポートへ届くだけでは制御できず、鍵が
要ります (SSH は無関係)。鍵は `rndc-confgen` で作ります。

1-4-2 書式ブロック

`rndc` :稼働中の `named` へ制御コマンドを送る

書式 `rndc [-c 設定ファイル] [-s サーバー] [-p ポート] [-y 鍵名] <サブ
コマンド> [引数]`

例 `rndc reload example.com`

例 `rndc retransfer example.com`

サブコマンド	何をするか
<code>reload</code>	<code>named.conf</code> と、 変更されたゾーンファイル も読み直す (ゾーン名でそれだけ)
<code>reconfig</code>	<code>named.conf</code> を読み直し、 新規ゾーンだけ 取り込む (既存ゾーンは読み直さない)
<code>retransfer</code>	セカンダリ で、指定ゾーンをマスターから強制的に全転送し直す
<code>refresh</code>	マスターへの更新確認を今すぐ行う
<code>flush</code>	キャッシュを破棄する (<code>flushname</code> で特定の 名前だけ)
<code>freeze</code> / <code>thaw</code>	動的更新を止めて手編集できるようにする ／再開する
<code>stop</code> / <code>halt</code>	更新を保存して停止／保存せず即停止

1-4-3 reload と reconfig と retransfer の使い分け

やったこと	使うコマンド
<code>named.conf</code> に新しいゾーンを追記しただけ	<code>rndc reconfig</code>
新規ゾーンの追加と、既存ゾーンファイルの編集の両方	<code>rndc reload</code>
1つのゾーンファイルだけを編集した	<code>rndc reload <ゾーン名></code>
セカンダリで、マスターの内容を今すぐ取り直したい	<code>rndc retransfer <ゾーン名></code>
キャッシュに残った古い答えを消したい	<code>rndc flush</code>

`reconfig` は、既存ゾーンのファイル確認そのものを省きます。そのため、編集した既存ゾーンは反映されません。`reload` は、変更の有無を確かめてから変わったものを読み直します。`reload` はローカルのファイルを読み、`retransfer` はマスターから転送し直してもらいます。`dig` で AXFR を投げて、表示されるだけでセカンダリのデータは更新されません。`flush` は、設定もゾーンも読み直しません。

`kill -HUP <PID>` も `rndc reload` に相当します (SIGTERM は終了)。ただし、ゾーン単位の指示やリモート操作はできません。

1-4-4 rndc status の出力を読む

```
$ rndc status
version: BIND 9.18.49-1~deb12u2-Debian (Extended Support Version) <id:>
number of zones: 7 (0 automatic)
debug level: 0
xfers running: 0
xfers deferred: 0
soa queries in progress: 0
query logging is OFF
recursive clients: 0/900/1000
tcp clients: 0/150
server is up and running
```

行	読み方
server is up and running	named が動作中で、rndc の制御チャンネルに応答している
query logging is OFF	問い合わせのログ出力は 無効 (rndc querylog on で有効にする)
xfers running / xfers deferred	実行中のゾーン転送の数 / 実行待ちの数。ともに 0 なら転送は動いていない
recursive clients: 0/900/1000	処理中の再帰問い合わせの数 / 上限の目安。0 は「今は処理していない」で、再帰が無効という意味ではない
debug level: 0	デバッグ出力は無し。デバッグ情報がログに出ているわけではない

1-4-5 編集したゾーンを反映する: reload・SIGHUP と、反映されない例

プライマリでゾーンファイルに A レコードを 1 行足し、SOA のシリアルも増やしたとします。

```
# rndc reconfig  
# host smtp.example.jp 127.0.0.1  
Host smtp.example.jp not found: 3(NXDOMAIN)
```

`rndc reconfig` は `named.conf` を読み直し、新しく追加されたゾーンを読み込み、削除されたゾーンを取り外すだけです。**すでに読み込んでいるゾーンのファイルは読み直しません。**もう一度 `reconfig` を実行しても同じです。ゾーンファイルの変更を反映するのは `rndc reload <ゾーン名>` (ゾーン名を省くと全ゾーン) です。`zone reload queued` と出て、1 秒ほどで新しい名前が引けるようになります。`rndc flush` はキャッシュを消すだけで権威ゾーンのデータには影響しません。`named-checkzone` は構文を検査するだけで `named` には何も伝えません。

named を停止せずに反映する別の方法が `kill -HUP $(pidof named)` です。`named` は SIGHUP を受け取ると設定ファイルとゾーンファイルを読み直します (`rndc reload` と同じ働き)。鍵が合わないなど `rndc` が使えない場面の代わりになります。`kill -TERM` は `named` を終了させる信号だ。

シリアルを増やし忘れても、プライマリでは `reload` でファイルは読み直されます。ただしセカンダリは SOA のシリアルが増えていないとゾーンを取得しないので、変更時はシリアルを必ず増やします。新しいゾーンを足しただけの場合は `rndc reconfig` で足ります。

`rndc-confgen -a` は、鍵ファイルを直接書き出します。 `-a` なしの `rndc-confgen` は、`rndc.conf` と `named.conf` に貼る設定の例を画面に出すだけで、

ファイルを作りません。`-a` を付けると、`wrote key file "/etc/bind/rndc.key"` と表示して、既定の場所に `rndc.key` を書きます。Debian では `/etc/bind/rndc.key` (所有者は `bind`)、Rocky 9 では `/etc/rndc.key` です。`-c` でファイルの場所を変えられます。`-k` は鍵の名前を決めるだけで、ファイルへは書きません。生成後は `named` を再起動して、鍵を読み込ませます。

1-5 設定を壊さないための検証コマンド [207.1/207.2]

1-5-1 書式ブロック

named-checkconf : `named.conf` の構文を検査する

書式 `named-checkconf [-z] [-p] [-t ディレクトリ] [設定ファイル]`

既定では `named.conf` 自体の構文しか見ません。`-z` を付けると、マスターとして定義された全ゾーンのファイルも読み込んで検証します。問題が無ければ、何も出力しません。

named-checkzone : ゾーンファイルの構文と整合性を検査する

書式 `named-checkzone [オプション] <ゾーン名> <ゾーンファイル>`

例 `named-checkzone example.com /var/named/example.com.zone`

引数は、**ゾーン名が先、ファイルが後**です。ゾーン名を省いた形や、`--zone` のような長い書式はありません。成功すると `zone example.com/IN: loaded serial 2026010101` と `OK` を表示します (checkconf と違い、成功時も出力がある)。

named-compilezone : ゾーンファイルの形式を変換する

書式 `named-compilezone [-f 入力形式] [-F 出力形式] -o <出力ファイル> <ゾーン名> <ゾーンファイル>`

例 `named-compilezone -f text -F raw -o example.com.db example.com
/var/named/example.com.zone`

`named-checkzone` と同じ検証をする兄弟です。どちらも `-f` が**入力形式**、`-F` が**出力形式**で、`text`・`raw`・`map` を扱えます。違いは `-o` だけです。`checkzone` では省略でき(検証だけ)、`compilezone` では必須です(変換結果を書き出す)。テキストから生形式へは `-f text -F raw`、戻すときは `-f raw -F text` で、相互変換できます。

1-5-2 named-checkzone の警告の読み方

`named-checkzone` は、最後に `OK` と出ても、途中で警告を出すことがあります。**警告は読み飛ばさず、直します。**

```
$ named-checkzone example.jp db.example.jp
zone example.jp/IN: example.jp/MX 'mail.example.jp' is a CNAME
(illegal)
zone example.jp/IN: loaded serial 2026101103
OK
```

MX の指す先は、A または AAAA レコードを持つホスト名でなければなりません。CNAME を指すと `is a CNAME (illegal)` と警告されます。直し方は、MX の指す先を実体のホスト名(上の例なら `www.example.jp.`)にするか、`mail` を CNAME ではなく A レコードにすることです。同じ名前に CNAME と TXT を併

記する案は、CNAME が他のレコードと同居できないので別のエラーになります。MX の優先度は MX 同士の順序を決める値で、TTL とは無関係だ。

委任先のネームサーバーの名前が、委任したゾーン自身の中にある場合は、親ゾーンに glue レコード(A/AAAA)が要ります。

```
dev      IN  NS  ns1.dev.example.jp.
```

```
dev.example.jp/NS 'ns1.dev.example.jp' has no REQUIRED GLUE address
records (A or AAAA)
```

ns1.dev.example.jp のアドレスを知るには、dev.example.jp の委任先に問い合わせる必要があります。問い合わせるにはそのアドレスが要る、という循環が起きます。これを断つために、親ゾーンに ns1.dev IN A 192.0.2.54 のように、委任先 NS 名のアドレスを書きます。それが glue だ。dev.example.jp 配下のほかのホストの記録は子ゾーンの管理範囲なので、親ゾーンに書いても応答に使われません。NS の TTL を延ばしても、SOA の refresh(セカンダリが更新を確認する間隔)を縮めても、アドレスが無い問題は解消しません。

1-5-3 named-checkconf -z : ゾーンファイルも読み込んで検査する

named-checkconf は、既定では named.conf の構文だけを検査します。マスター(プライマリ)として定義されたゾーンのファイルが存在しなくても、何も言わず終了ステータス 0 で終わります。

-z を付けると、プライマリとして定義された**各ゾーンのファイルも読み込み**、ゾーンとして成立するか検査します。

```
$ named-checkconf -z; echo $?  
zone example.jp/IN: loaded serial 2026101101  
zone y.jp/IN: loading from master file /nonexistent failed: file not  
found  
zone y.jp/IN: not loaded due to errors.  
1
```

読み込んだゾーンは `loaded serial ...` と表示され、ファイルが無いなどで失敗したゾーンがあると、終了ステータスが 1 になります。稼働中の `named` に接続して差分を調べる、セカンダリでプライマリからの転送を試す、書式を厳しくする、といった動作ではありません。

1-6 ゾーンファイルの構造 [207.2]

```
$TTL 86400
@      IN SOA  ns1.example.jp. root.example.jp. (
                                2026010101      ; Serial
                                3600              ; Refresh
                                900               ; Retry
                                604800           ; Expire
                                86400 )          ; Minimum (negative cache TTL)

@      IN NS   ns1.example.jp.
@      IN MX   10 mail1.example.jp.
@      IN MX   20 mail2.example.jp.
ns1     IN A    192.0.2.53
web1    IN A    192.0.2.10
www     IN CNAME web1
ipv6host IN AAAA 2001:db8::10
@      IN TXT  "v=spf1 mx -all"
```

1-6-1 \$TTL と \$ORIGIN と @

\$ で始まる行は、レコードではなくディレクティブです。

- **\$TTL** : ファイル全体の**既定の TTL** (秒) です。個々のレコードに書いた TTL が優先されます (上書きされない)。DNSSEC 署名の期限とは無関係です
- **\$ORIGIN** : **末尾にドットの無い名前へ補うドメイン名** (原点) を切り替えます。原点は `zone` 文のゾーン名から始まります

- **@** : その時点の原点そのものを表します (名前欄を空白にすると、直前の名前が引き継がれる)

末尾のドットが決定的です。ドットで終わる名前は完全修飾名で、原点は補われません。付け忘れると、`ns1.example.jp.example.jp.` のような二重の名前になります。

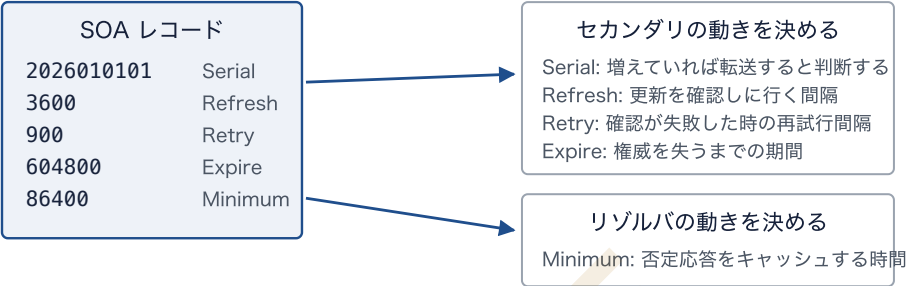
ディレクティブ	働き
<code>\$INCLUDE <ファイル> [<原点>]</code>	別のファイルをその位置へ読み込む
<code>\$GENERATE <範囲> <名前> <タイプ> <データ></code>	番号だけ違うレコードを連番で作る (BIND 独自の拡張)

1-6-2 SOA レコードの 5 つの時間パラメータ

パラメータ	効き先	意味
Serial	セカンダリ	ゾーンの世代番号 (慣習で <code>YYYYMMDDnn</code>)。増えていれば転送する
Refresh	セカンダリ	マスターへ更新を確認しに行く間隔
Retry	セカンダリ	確認に失敗した時の再試行間隔
Expire	セカンダリ	マスターと通じないまま、権威ある応答をやめるまでの期間
Minimum	リゾルバ	ネガティブキャッシュ TTL (否定応答をキャッシュする時間。RFC 2308)

Retry を「データを破棄するまでの時間」とする肢は、Expire と混せています。

SOA の 5 つの値 — それぞれ誰のどの動作に効くか



Serial は世代を表す番号で、時間パラメータではない(慣習として YYYYMMDDnn の形)。
Minimum だけは向いている先が違い、ネガティブキャッシュ TTL を表す。ゾーンデータの失効期限でも更新確認の間隔でもない。
\$TTL は、レコードで TTL を明示していない時に使われる既定 TTL。SOA の 5 つとは別の指定。

図 1-3 SOA の 5 つの値:Minimum だけリゾルバに効く

1-6-3 リソースレコードの種類

タイプ	対応づけるもの
A / AAAA	ホスト名 → IPv4 / IPv6 アドレス
CNAME	別名 → 正式なホスト名
MX	ドメイン → メールを受け取るサーバー (優先度付き)
NS	ゾーン → 受け持つネームサーバー
PTR	IP アドレスから作った名前 → ホスト名 (逆引き)
SOA	ゾーンの管理情報 (1ゾーンに1つ)
TXT	任意のテキスト (SPF など)

タイプ	対応づけるもの
DNSKEY ・ RRSIG ・ DS ・ NSEC / NSEC3	DNSSEC 用 (→ 1-13)
TLSA	DANE 用 (→ 1-14)

1-6-4 MX レコードの優先度

MX の数値は優先度で、**値が小さいほど優先**です(帯域や性能の値ではなく、連番でなくてよい)。

1-6-5 CNAME の制約

- CNAME を付けた名前には、同じ名前で `MX` や `A` など他のレコードを併記できません
- 同じ名前に、複数の CNAME は置けません

MX が指す先に、CNAME を使うのは避けます。

ゾーンファイルの 1 行は、名前 TTL クラス 種別 データの順に書きます。TTL とクラスは入れ替えてもよい。`1h` のように単位をつけた TTL も使える。次の 3 行は、どれも `web1.example.jp. 3600 IN A 192.0.2.10` として読み込まれます。

```
web1 3600 IN A 192.0.2.10
web1 IN 3600 A 192.0.2.10
web1 1h IN A 192.0.2.10
```

TTL を種別のあとへ置くと、`3600` がアドレスとして読まれます。`named-checkzone` は `bad dotted quad` で読み込みに失敗します。名前をドット付きの

web1.example.jp. で書くなら、完全修飾名になる。ドットなしの web1.example.jp は相対名で、web1.example.jp.example.jp. になります。

\$GENERATE 範囲 左辺 種別 右辺 は、範囲の数値を \$ の位置に入れて、行を展開します。名前の側もデータの側も置き換わる。

```
$GENERATE 1-3 pc$ A 192.0.2.$
```

named-checkzone -D で展開結果を出すと、pc1 が 192.0.2.1、pc2 が 192.0.2.2、pc3 が 192.0.2.3 の A レコードになる。1 つの名前に 3 つの A がつくのではありません。\$ は 3 つの名前を作り、データも 1、2、3 と変わる。種別は A に限らず、逆引きの PTR (\$GENERATE 1-254 \$ PTR host\$.example.jp.) や CNAME にも使えます。

ワイルドカードは、名前の欄に * を書きます。使われるのは、**問い合わせた名前そのものがゾーンに無いとき**だけです。

```
www IN A 192.0.2.10
*    IN A 192.0.2.99
```

問い合わせ	結果	理由
mail.example.jp の A	NOERROR • 192.0.2.99	名前が無いので * が使われる
www.example.jp の A	NOERROR • 192.0.2.10	名前があるので、その名前のレコードが優先される
www.example.jp の AAAA	NOERROR • 回答なし (NODATA)	名前はあるので、種別が違って * は代わりにならない

問い合わせ	結果	理由
x.www.example.jp のA	NXDOMAIN	www が存在するため、その下には * が届かない

「ワイルドカードはAにしか使えない」という説明は誤りです。種別を問わず使える。AAAA が空なのは、www という名前が存在するからです。

1-7 逆引きゾーン [207.2]

逆引きは、正引きとは別のゾーンです。片方に登録しただけでは、もう片方は引けません。

1-7-1 ゾーン名の作り方

ネットワーク部のオクテットを逆順に並べ、in-addr.arpa を付けます。
192.0.2.0/24 なら 2.0.192.in-addr.arpa です (10.0.0.0/8 なら 10.in-addr.arpa)。逆順にしない形や、ホスト部の 0 を含めた形は誤りです。IPv6 は ip6.arpa で、4ビットごとに区切って逆順に並べます。

1-7-2 PTR レコード

@	IN NS	ns1.example.jp.
10	IN PTR	web1.example.jp.
25	IN PTR	mail1.example.jp.

左にはホスト部のオクテット、右には末尾にドットを付けた完全修飾のホスト名を書きます。ドットを忘れると原点が補われ、意味のない名前になります。

1-7-3 ホストを 1 台足す時の手順

- 1. 正引きのゾーンファイルに `web1 IN A 192.0.2.10` を追加する
- 2. 逆引きのゾーンファイルに `10 IN PTR web1.example.jp.` を追加する
- 3. 両方のゾーンファイルで SOA のシリアル番号を増やす
- 4. `named-checkzone` で検証してから `rndc reload` で反映する

1-8 委任とグルーレコード [207.2]

サブドメインの管理を別サーバーへ任せる**委任**は、**親ゾーンに子ゾーンの NS レコードを置く**ことで表します。

sales	IN NS	ns.sales.example.com.	
ns.sales	IN A	192.0.2.60	; グルーレコード

委任先のサーバー名が子ゾーンの内部にあると、そのアドレスを引くのに子ゾーンが要り、堂々巡りになります。そこで、親に委任先の A レコードを添えます。これが**グルーレコード**です。委任先の名前が子ゾーンの外なら、グルーは要りません。SOA は各ゾーンが自分で持つもので、CNAME は委任の意味を持ちません。

1-9 マスターとセカンダリ: ゾーン転送のしくみ [207.2]

1-9-1 転送が起きるきっかけ

きっかけ	動き
定期確認	セカンダリが Refresh 間隔で SOA を問い合わせ、シリアルを比べる

きっかけ	動き
NOTIFY	マスターが更新時に通知し、Refresh を待たずに確認させる
手動	<code>rndc retransfer <ゾーン名></code>

セカンダリは、マスターのシリアルが自分の値より大きい時だけ転送します。Refresh ごとに無条件で AXFR するわけではありません。

ゾーンファイルを編集したら、必ずシリアル番号を増やします。増やし忘れると転送がスキップされ、**セカンダリだけが古いデータを返し続けます**。構文エラーではないので、マスターは何事もなく動き続けます（エラーログも SERVFAIL も出ない）。

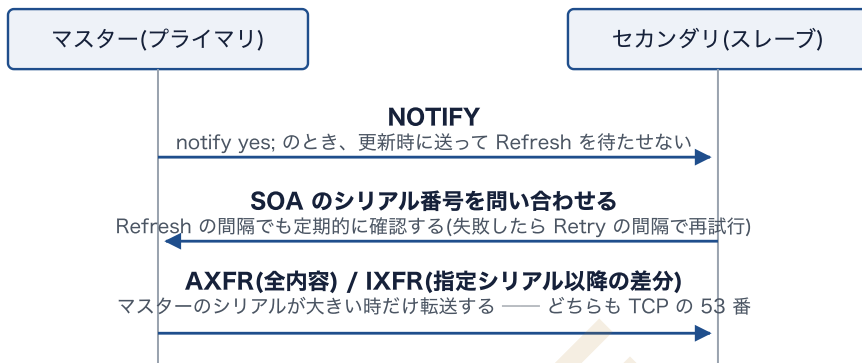
1-9-2 NOTIFY

`notify yes;`（既定）なら、マスターは更新時に NS レコードのセカンダリへ NOTIFY を送ります。NS に載らない相手へは `also-notify` に並べます。速く変更を伝える答えは `notify` です。`allow-transfer { none; };` では転送そのものが止まり、Refresh を `0` にしても即時通知にはなりません。

1-9-3 AXFR と IXFR

AXFR はゾーンの全内容、**IXFR** は指定シリアル以降の差分だけを転送します。どちらも TCP の 53 番を使います。

ゾーン転送 —— 転送するかどうかはシリアル番号で決まる



ゾーンファイルを編集してシリアルを増やし忘れると、セカンダリからは「変更なし」としか見えず、転送がスキップされて古いデータが残り続ける。マスター側はエラーにならず動き続けるので、症状は「セカンダリだけが古いまま」という形で現れる。
allow-transfer で相手を絞らないと、ゾーン内のホスト一覧が第三者に丸ごと渡る。

図 1-4 ゾーン転送:NOTIFY・シリアル比較・AXFR と IXFR

1-9-4 DNS が使うポート:UDP と TCP の両方の 53 番

DNS は短い問い合わせと応答を **UDP の 53 番**で行う。次の場合は **TCP の 53 番**が使われます。

- 応答が大きくて UDP に収まらず、切り詰められた(TC ビットが立った)時。
クライアントは TCP で問い合わせ直す
- **ゾーン転送(AXFR/IXFR)**

したがって、権威サーバーの前のファイアウォールでは **UDP と TCP の両方の 53 番**を通します。UDP だけだと、大きな応答とゾーン転送がエラーになります。**953 番(TCP)**は **rndc** が **named** を操作する制御用のポートで、DNS の問い合わせやゾーン転送には使いません。UDP の 953 番は DNS と無関係です。

1-9-5 ゾーン転送が失敗する原因: dig AXFR で確かめる

`dig @サーバー ゾーン名 AXFR` は、ゾーン全体を一度に受け取る操作です。失敗すると、`; Transfer failed.` と出ます。原因は次の 3 つに分かれる。

原因	見立て
プライマリの <code>allow-transfer</code> が、問い合わせ元のアドレスを許可していない	拒否される。許可は <code>allow-transfer { 192.0.2.2; };</code> のように書く
TCP の 53 番 をファイアウォールが遮断している	転送は TCP なので届かない
問い合わせ先の <code>named</code> が、 そのゾーンを持っていない	転送元になれない

次は原因になりません。UDP の 53 番が閉じていても、転送は TCP なので成否に直接関係しません。SOA の `serial` の大小も関係しません。`dig` の AXFR は `serial` を比せず、いつでもゾーン全体を要求します。`serial` の大小は、セカンダリが更新するかどうかを決める NOTIFY や IXFR の判断に使う値です。

1-9-6 セカンダリが更新されないとき: REFUSED と `primary • secondary` の呼び名

セカンダリ側の `named` のログに次のように出ているときは、プライマリが転送を断っています。

```
xfer-in: error: transfer of 'example.jp/IN' from 172.17.0.4#53: failed
while receiving responses: REFUSED

xfer-in: info: transfer of 'example.jp/IN' from 172.17.0.4#53: Transfer
status: REFUSED
```

接続自体はできています (`connected using ...` が出ている)。原因は、プライマリの `allow-transfer` が**実際のセカンダリのアドレスを許可していない**ことです。プライマリの設定が `allow-transfer { 172.17.0.20; };` だとします。セカンダリが 172.17.0.10 なら、許可するアドレスを 172.17.0.10 に直します。ファイアウォールが TCP の 53 番を遮断している場合は、`REFUSED` ではなく接続できない形で現れます。SOA の serial を増やす必要があるのは、更新を検知させるためで、転送が拒否される原因ではありません。許可は、セカンダリ側ではなく**プライマリ側**に書きます。

BIND 9.16 以降は、`type master` ・ `type slave` ・ `masters` の代わりに `type primary` ・ `type secondary` ・ `primaries` と書けます。意味は同じで、古い書き方も使えます。

1-10 問い合わせの道具：`dig` ・ `host` ・ `nslookup` [207.2]

1-10-1 三つの道具の位置づけ

コマンド	特徴
<code>dig</code>	応答の全体 (ヘッダー・フラグ・各セクション) を詳しく表示する
<code>host</code>	出力が簡潔。ホスト名なら正引き、IP アドレスなら逆引き
<code>nslookup</code>	対話モードを持つ (一時の非推奨は撤回され、今も BIND に同梱)

dig :DNS への問い合わせと応答の詳細表示

書式 dig [@サーバー] [オプション] <名前> [レコードタイプ] [+クエリオプション]

例 dig example.com A

例 dig @192.0.2.53 example.com A

例 dig example.com MX

例 dig +short example.com A

例 dig example.com +trace 例 dig -x 203.0.113.10

例 dig @192.0.2.53 example.com AXFR

オプション	意味
@<サーバー>	問い合わせ先を指定し、そのサーバーへ直接問い合わせる(省略時は /etc/resolv.conf)
-x <IP>	逆引き。in-addr.arpa / ip6.arpa 形式へ自動変換して PTR を引く
+short	結果の値だけを表示する
+trace	ルートから順にたどる反復解決の過程をすべて表示する

問い合わせ先は @ の書式だけです (-s や --server= は存在しない)。タイプとクエリオプションの位置は自由で、タイプを省くと A を引きます。-x は逆引き専用です (-R や -PTR は存在しない)。dig 203.0.113.10 PTR と書いても、in-addr.arpa へ変換されません。+short は、表示を絞るだけです。

ゾーン転送の確認は dig @<サーバー> <ゾーン名> AXFR です。許可されていれば全レコードが返り、制限されていれば Transfer failed. になります。

host : 簡潔な名前解決

書式 `host [-t タイプ] [-a] [-l ゾーン] [-C] <名前 | IP アドレス> [サーバー]`

例 `host www.example.jp` 例 `host -t MX example.jp`

`-a` は、取得できる全種類のレコードを表示します (`-t ANY -v` 相当)。

nslookup : 対話モードを持つ問い合わせコマンド

書式 `nslookup [-type=タイプ] [名前] [サーバー]`

例 `nslookup www.example.jp` 例 `nslookup -type=MX example.jp`

引数なしで対話モードになります(プロンプトで続けて問える)。`server <アドレス>` で問い合わせ先、`set type=MX` (`set q=` も同じ) でタイプを切り替え、`exit` で終わります。

1-10-2 dig の status を読む: NXDOMAIN • NOERROR • SERVFAIL • REFUSED

`dig` の応答の `;; ->HEADER<<-` の行に出る `status:` が、結果の種類を表す。

status	意味
NOERROR	問い合わせに成功した。 <code>ANSWER: 0</code> なら、名前は存在するが、その種類のレコードが無い
NXDOMAIN	その名前が存在しないと、問い合わせ先が答えた

status	意味
SERVFAIL	問い合わせ先がうまく処理できなかった (DNSSEC の検証失敗など)
REFUSED	問い合わせ先が、アクセス制御などでその 問い合わせに答えるのを断った(ゾーン転送 が許可されていないときもこれ)

```
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 17520
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; AUTHORITY SECTION:
example.jp. 300 IN SOA ns1.example.jp. hostmaster.example.jp.
2026101101 3600 900 604800 300
```

flags の aa は**権威のある応答** (Authoritative Answer) の印です。この出力は、example.jp の権威サーバーが「その名前は存在しない」と答えたことを表す。AUTHORITY セクションの SOA は、否定の応答をキャッシュしてよい時間 (SOA の TTL と最後の値のうち小さい方) を伝えています。アクセス制御で拒否された場合は REFUSED になります。名前が存在するのにレコードが無い場合は NOERROR で ANSWER: 0 になります。

1-10-3 host に IP アドレスを渡すと逆引きになる

host に IP アドレスを渡すと、自動的に**逆引き (PTR レコードの問い合わせ)** になります。-t PTR を付けなくても同じだ。

```
$ host www.example.jp 127.0.0.1
www.example.jp has address 192.0.2.80
$ host 192.0.2.80 127.0.0.1
Host 80.2.0.192.in-addr.arpa. not found: 3(NXDOMAIN)
```

192.0.2.80 は `80.2.0.192.in-addr.arpa` という名前に変換されます。この名前
で問い合わせられます(エラーメッセージの名前が `in-addr.arpa` になってい
るのがその証拠です)。正引きは `example.jp` のゾーンにあり、逆引きは別の逆
引きゾーン(`2.0.192.in-addr.arpa`)で管理します。そのため、**正引きが引けて
も逆引きが引けるとは限りません**。同じサーバーが正引きに答えているので、
`named` は停止していません。

原因は、逆引きゾーンが無いが、`80 IN PTR www.example.jp.` の PTR レコード
が無いことです。逆引きゾーン(`zone "2.0.192.in-addr.arpa"`)とゾーンファイ
ルを追加して `rndc reconfig` します。すると、`80.2.0.192.in-addr.arpa domain
name pointer www.example.jp.` が返ります(新しいゾーンの追加は `reconfig`
で足ります)。PTR を登録していない `192.0.2.81` は `NXDOMAIN` のままです。A
レコードを足しても逆引きにはなりません。

1-11 アクセス制御と情報の隠蔽 [207.3]

1-11-1 acl でアドレスの集合に名前を付ける

```
acl "internal" { 127.0.0.1; 192.0.2.0/24; 198.51.100.0/24; };
options {
    allow-query      { any; };
    allow-recursion { "internal"; };
    allow-transfer   { none; };
};
```

組み込みの名前は、**any**（すべて）、**none**（なし）、**localhost**（自分のアドレス）、**localnets**（自分が属するネットワーク）です。**options** ブロックは **named** 全体に効くグローバルな設定の置き場で、ゾーンファイルの置き場所もここに書きます（**directory**）。

ディレクティブ	何を制限するか
<code>allow-query</code>	通常の問い合わせ
<code>allow-recursion</code>	再帰問い合わせ
<code>allow-transfer</code>	ゾーン転送
<code>allow-update</code>	動的更新（ <code>key</code> 名も書ける）

権威サーバーとして公開するなら、`allow-query { any; }` で開きます。そのうえで、`recursion no;` か `allow-recursion` を内部に絞ります。再帰を誰にでも許すと、増幅攻撃の踏み台になりかねません。

1-11-2 版の隠蔽と view

`version "unknown";` は、`version.bind` に実際の版を返さないための措置です。`view` は、問い合わせ元で見せるゾーンを切り替えます (split DNS)。`view` を1つでも書いたら、すべての `zone` 文をいずれかの `view` に入れます。

1-11-3 chroot 環境での稼働

```
書式  named -t <chroot のルート> -u <実行ユーザー> [-c 設定ファイル] [-g]
```

例 `named -t /var/named/chroot -u named`

`-t` は chroot のルート、`-u` は起動後に権限を落とすユーザーです。

chroot 先には、稼働に必要なファイル一式を置き直します。`named.conf`、ゾーンファイル、共有ライブラリ、`/dev/null` や `/dev/random`、`/etc/localtime` などです (外の `/usr/lib` や `/etc` は自動では参照されない)。`-u` のユーザーは、chroot 先の `/etc/passwd` に無くても構いません。`named` はルートを切り替える前に本来の `/etc/passwd` を引き、UID と GID を確定させるからです。

chroot は、通信の暗号化や改ざん検知をしません (それは TSIG など)。

1-11-4 chroot の中に置くもの: `root.hints` も chroot の下に

`named -t` ディレクトリ は、起動後にそのディレクトリを `/` として扱う chroot の指定です。`named` が開くファイルは、`-c` で指定する設定ファイル、ゾーンファイル、hint ゾーンのファイルなど、**すべて chroot の下から見なければなりません**。`chroot` の外側のパスは、設定に書いても参照できません。

```
# named -u bind -t /var/chroot-named -c /etc/bind/named.conf
could not configure root hints from '/usr/share/dns/root.hints': file
not found

loading configuration: file not found

exiting (due to fatal error)
```

このエラーは、chroot の下の `usr/share/dns/root.hints` が無いために設定の読み込みが失敗した例です。`/var/chroot-named/usr/share/dns/root.hints` にファイルをコピーすれば起動します。`-t` は設定ファイルの場所を指定するオプションではなく、`-c` を外す必要もありません。`-u` は起動後の実行ユーザーを指定するもので、chroot とは別です (外せば解決するわけではありません)。`dev/null`・`dev/random`・`dev/urandom` なども `mknod` で作って置きます。

RHEL 系では、`bind-chroot` を入れると `/var/named/chroot` が整えられ、`named-chroot.service` で起動する。

1-12 TSIG:通信相手を共有鍵で認証する [207.3]

TSIG (Transaction Signatures) は、共有した秘密鍵で個々の DNS メッセージに HMAC を付ける仕組みです。主にゾーン転送と動的更新で、通信相手が正しいことと、改ざんが無いことを確かめます。

1-12-1 DNSSEC との違い

	TSIG	DNSSEC
鍵の方式	共有秘密鍵 (対称鍵) + HMAC	公開鍵と秘密鍵のペア
守る対象	個々の通信	ゾーンのレコードそのもの

	TSIG	DNSSEC
主な用途	ゾーン転送・動的更新の相手認証	リゾルバによる応答の検証
設定の場所	named.conf の key 文と server 文	鍵ファイルと署名済みゾーンファイル

公開鍵でゾーンに署名して RRSIG を作るのは、DNSSEC (dnssec-signzone) です。TSIG は通信内容を暗号化しません(それは DoT / DoH)。送信元アドレスだけで判断する allow-transfer の制限とも別物です。

1-12-2 設定の流れ

```
key "example-key" { algorithm hmac-sha256; secret
"K2c3ZXhhbXBsZS1rZXktc2VjcmV0LXZhbnVl"; };
server 192.0.2.54 { keys { "example-key"; }; };
zone "example.jp" IN { type master; file "example.jp.zone"; allow-
transfer { key "example-key"; }; };
```

- 1. key で鍵の名前、アルゴリズム、秘密鍵の値を定義する
- 2. server で相手に使う鍵を指定するか、allow-transfer / allow-update に key <鍵名> を書く
- 3. 同じ鍵をマスターとセカンダリの両方に置く

鍵はゾーンファイルではなく、named.conf に書きます。

tsig-keygen : TSIG 用の共有鍵を生成する

書式 `tsig-keygen [-a アルゴリズム] <鍵名>`

例 `tsig-keygen example-key`

`named.conf` へそのまま貼れる `key { }` 形式で出力します (既定は `hmac-sha256`)。 `dnssec-keygen` は DNSSEC 用、 `rndc-confgen` は `rndc` 用です。

1-12-3 allow-transfer に鍵を書く: `tsig-keygen` と `dig -k`

鍵は `tsig-keygen` (古い手順では `dnssec-keygen`) で作り、 `named.conf` に `key` 文として取り込みます。

```
$ tsig-keygen -a hmac-sha256 xfr-key
key "xfr-key" {
    algorithm hmac-sha256;
    secret "...";
};
```

出力を鍵ファイル (例: `/etc/bind/xfr.key`。所有者 `root:bind`・モード `640`) に保存して `include` し、ゾーン転送の許可に `key 鍵名;` を書きます。

```
zone "example.jp" { type master; file "db.example.jp"; allow-transfer {
key xfr-key; }; };
```

指定	制御するもの
<code>allow-transfer</code>	ゾーン転送(アドレスの代わりに <code>key</code> 鍵名; と書くと、その鍵で署名した要求だけ許可する)
<code>allow-query</code>	通常の問い合わせ
<code>allow-update</code>	動的更新
<code>allow-recursion</code>	再帰問い合わせ

確かめるには、鍵なしと鍵付きの `dig` を比べるとよい。鍵を付けない `dig @127.0.0.1 example.jp AXFR` は `Transfer failed.` になります。`dig -k /etc/bind/xfr.key @127.0.0.1 example.jp AXFR` を使うと、SOA・NS・MX などが転送されます。名前の違う鍵を使うと、応答の TSIG に `BADKEY` が付いて `Transfer failed.` になります。

1-13 DNSSEC: 応答の正当性を検証できるようにする [207.3]

DNSSEC は、レコードセットに公開鍵暗号で署名し、リゾルバが本物か検証できるようにします。改ざんが無いことの証明で、暗号化ではありません。

1-13-1 使われるレコード

レコード	役割
<code>DNSKEY</code>	ゾーンの公開鍵 (ZSK と KSK の両方)
<code>RRSIG</code>	各レコードセットへの電子署名
<code>DS</code>	子ゾーンの KSK のハッシュ。親に置き、信頼の連鎖をつなぐ

レコード	役割
NSEC / NSEC3	名前が存在しないことの証明 (NSEC3 はハッシュ化して列挙を防ぐ)

1-13-2 二種類の鍵

鍵	何に署名するか
ZSK (ゾーン署名鍵)	ゾーン内の実際のレコード。既定で生成されるのはこちら
KSK (鍵署名鍵)	DNSKEY レコード。この鍵のハッシュが親に DS として登録される

1-13-3 導入の手順と書式ブロック

手順は、① `dnssec-keygen` で鍵を作る → ② `dnssec-signzone` で署名する → ③ 親へ DS を登録するです。

`dnssec-keygen` : DNSSEC 用の鍵ペアを生成する

書式 `dnssec-keygen [-a アルゴリズム] [-b 鍵長] [-f KSK] [-n ZONE] [-K ディレクトリ] <ゾーン名>`

例 `dnssec-keygen -a RSASHA256 -b 2048 example.com`

例 `dnssec-keygen -f KSK -a RSASHA256 -b 2048 example.com`

`K` で始まる公開鍵 (`.key`) と秘密鍵 (`.private`) の組が、カレントディレクトリにできます (`-K` で変更)。`-f KSK` の有無だけが ZSK と KSK を分けます (`-t` は鍵の型の別オプション)。

dnssec-signzone : ゾーンファイルに署名する

書式 `dnssec-signzone [-o オリジン] [-N 方式] [-K 鍵ディレクトリ] <ゾーンファイル> [鍵ファイル...]`

例 `dnssec-signzone -o example.com /var/named/example.com.zone`

オプション	意味
<code>-o <オリジン></code>	ゾーン名を明示する (省略時はファイル名から推測)
<code>-k <ファイル></code>	KSK を明示する
<code>-N <方式></code>	シリアルの扱い(<code>increment</code> ・ <code>unixtime</code> ・ <code>keep</code>)
<code>-3 <salt></code>	NSEC3 を使う
<code>-f <ファイル></code>	出力ファイル名

出力は通常「元のファイル名 + `.signed`」で、これを `zone` 文の `file` に指定します。親に登録する DS をまとめた `dsset-<ゾーン名>` も作られます。**DS の親への登録は自動では行われません**(レジストラなどで手作業)。ファイル名とゾーン名が一致しない環境では、`-o` を明示します。

1-13-4 信頼の連鎖と検証側の設定

リゾルバは、ルートから「親の DS」と「子の DNSKEY」を照合して下り、最後に目的のレコードの RRSIG を検証します。これを**信頼の連鎖**と呼びます(どこかで切れると検証できない)。キャッシュサーバーで検証するには、`options` に `dnssec-validation auto;` を書きます。組み込みのルートの信頼点を使い、検証に失敗した応答は **SERVFAIL** になります。

DNSSEC の信頼の連鎖 —— 親の DS と子の DNSKEY を照合しながら下りる



図 1-5 DNSSEC の信頼の連鎖:親の DS と子の DNSKEY を照合しながら下りる

1-13-5 DNSSEC の検証失敗を切り分ける:+cd と RRSIG の期限

検証を有効にしたキャッシュサーバーで `SERVFAIL` が返った時は、`dig` に `+cd` (Checking Disabled、検証を行わない指定)を付けて、もう一度引きま

す。

```
$ dig @127.0.0.1 www.sign.test A → status: SERVFAIL
$ dig @127.0.0.1 www.sign.test A +cd → 応答が得られる
(flags に cd)
```

`+cd` で応答が得られるなら、データそのものは取得でき、失敗は DNSSEC の検証だけに限られます。次に、キャッシュサーバーのログで理由を見ます。

```
dnssec: info: validating sign.test/DNSKEY: verify failed due to bad
signature (keyid=30876): RRSIG has expired
```

`RRSIG has expired` は、署名(RRSIG)の**有効期限が切れた**ことを表す。署名には期限があり、期限が来る前に署名し直さなければなりません。根本的な対処は、`dnssec-signzone` で**署名し直して**ゾーンを読み込ませることです。検証を止める(`dnssec-validation no`)と応答は得られますが、改ざんを見抜くというDNSSECの目的が失われます。ログの原因は署名の期限切れで、DSの登録漏れではないので、DSを作り直しても直りません。`+cd` 付きで応答が得られているので、`allow-query` の問題でもありません。

DNSSECの鍵ファイル名は、K ゾーン名 + アルゴリズム番号 + キータグの形です。

```
$ dnssec-keygen -a ECDSAP256SHA256 -n ZONE example.jp
Kexample.jp.+013+23461
$ ls
Kexample.jp.+013+23461.key  Kexample.jp.+013+23461.private
```

`013` は**アルゴリズム番号**です。ECDSAP256SHA256 は 13 番で、3 桁に埋めて書かれます。`23461` は**キータグ**で、鍵ごとに付く識別子です。SOAのシリアル値でも、鍵の長さ、世代、有効期限の残りでもありません。`.key` が公開鍵(DNSKEYレコードの形)、`.private` が秘密鍵です。ZSKかKSKかは、ファイル名ではなく`.key`の中のコメントとフラグで区別します(ZSKの`.key`の先頭には `This is a zone-signing key` と書かれます)。`dnssec-signzone` で署名するときは、これらの鍵ファイルが同じディレクトリに要ります。

1-14 DANE と、問い合わせ自体を守る仕組み [207.3]

1-14-1 DANE と TLSA レコード

DANE (DNS-based Authentication of Named Entities) は、**TLS 証明書** の正当性を **DNS で検証できるようにする仕組み**です。**TLSA** レコードで、ホスト名とポートに証明書 (または公開鍵) を結び付けます。**DANE は DNSSEC が前提**です (TLSA が偽装されれば意味がない)。

1-14-2 DoT と DoH

名称	中身
DoT (DNS over TLS)	DNS を TLS で包む。 TCP の 853 番
DoH (DNS over HTTPS)	HTTPS に載せる。 443 番 で、Web 通信と区別しにくい

改ざんの検知は DNSSEC、転送相手の認証は TSIG、経路上の盗聴からの保護は DoT / DoH です。

1-15 章末のまとめ [207.1/207.2/207.3]

混同しやすい対の概念

対	違い
権威サーバーとキャッシュサーバー	自分のゾーンデータを持って答えるか、他所へ聞きに行き行って答えを預かるか
再帰問い合わせと反復問い合わせ	最終的な答えを丸投げするか、委任をたどって自分で聞き直すか

対	違い
<code>rd</code> と <code>ra</code>	問い合わせ側が再帰を要求したか、応答側が再帰に対応しているか
<code>aa</code> の有無	権威サーバーがゾーンデータから直接答えたか、キャッシュや再帰を経た非権威の答えか
<code>tc</code> フラグ	応答が切り詰められて TCP で再送されるか、そのまま UDP で収まったか
Unbound と NSD	キャッシュ (再帰) 専用か、権威専用か
dnsmasq と BIND	小規模向けで DHCP 機能も持つ軽量実装か、大規模運用にも耐える汎用実装か
<code>allow-query</code> と <code>allow-transfer</code>	通常の名前解決の問い合わせを制限するか、ゾーン転送を制限するか
<code>forward first</code> と <code>forward only</code>	転送先が答えない時に自分で解決しに行くか、そのまま失敗するか
<code>type master</code> と <code>type slave</code>	ゾーンファイルを自分で管理するか、転送で複製を受け取るか
<code>type hint</code> と <code>type forward</code>	ルートサーバーの一覧を持つゾーンか、問い合わせを別サーバーへ転送するゾーンか
<code>masterfile-format</code> と <code>masterfile-style</code>	ゾーンファイルの保存形式 (<code>text</code> / <code>raw</code>) か、書き出す時の名前の並べ方か
<code>rndc reload</code> と <code>rndc reconfig</code>	変更されたゾーンファイルも読み直すか、新規ゾーンの取り込みだけで既存は読み直さないか
<code>rndc reload</code> と <code>rndc retransfer</code>	ローカルのゾーンファイルを読み直すか、マスターから転送し直してもらうか

対	違い
<code>rndc reconfig</code> と <code>rndc flush</code>	<code>named.conf</code> を読み直すか、キャッシュを破棄するか
<code>named-checkconf</code> と <code>named-checkzone</code>	<code>named.conf</code> を検査するか、ゾーンファイルを検査するか
<code>named-checkconf</code> の <code>-z</code> の有無	参照先のゾーンファイルまで検証するか、設定ファイルの構文だけを見るか
<code>named-checkzone</code> と <code>named-compilezone</code>	<code>-o</code> を省略できて検証だけできるか、 <code>-o</code> が必須で変換結果を書き出すか
<code>-f</code> と <code>-F</code>	入力ファイルの形式か、出力ファイルの形式か
<code>\$TTL</code> と <code>\$ORIGIN</code>	既定のキャッシュ保持時間か、相対名へ補うドメイン名か
末尾のドットの有無	完全修飾名としてそのまま扱われるか、原点が補われるか
SOA の Refresh と Retry	定期確認の間隔か、確認に失敗した時の再試行間隔か
SOA の Retry と Expire	再試行の間隔か、権威を失うまでの総期間か
SOA の Serial と Minimum	ゾーンの世代番号か、否定応答のキャッシュ時間か
<code>A</code> と <code>AAAA</code>	IPv4 アドレスか、IPv6 アドレスか
<code>MX</code> と <code>NS</code>	メールの配送先か、ゾーンを受け持つネームサーバーか
MX の優先度 10 と 20	数値が小さい方が優先

対	違い
CNAME と A	別名を正式名へ対応づけるか、名前をアドレスへ対応づけるか
正引きゾーンと逆引きゾーン	A/AAAA を置く側か、PTR を置く側か (別のゾーンなので両方に登録する)
in-addr.arpa と ip6.arpa	IPv4 の逆引きか、IPv6 の逆引きか
NS レコードとグルーレコード	委任先のサーバー名を示すか、その名前のアドレスを親に添えるか
AXFR と IXFR	ゾーン全体の転送か、シリアル以降の差分転送か
NOTIFY と Refresh	マスターからの更新通知か、セカンダリからの定期確認か
dig と host と nslookup	詳細表示か、簡潔表示か、対話モードを持つか
dig @サーバー と -x	問い合わせ先の指定か、逆引きの指定か
+short と +trace	結果の値だけに絞るか、解決の過程をすべて表示するか
TSIG と DNSSEC	共有鍵で個々の通信を認証するか、公開鍵でレコードに署名するか
DNSSEC と DoT/DoH	改ざんを検知できるようにするか、経路上の盗聴から守るか
ZSK と KSK	実際のレコードに署名する鍵か、DNSKEY に署名し親に DS を登録する鍵か
dnssec-keygen と dnssec-signzone	鍵ペアを生成するか、その鍵でゾーンに署名するか

対	違い
tsig-keygen と rndc-confgen	ゾーン転送などに使う TSIG 鍵か、rndc の制御用の鍵か
DNSKEY と RRSIG と DS	公開鍵か、署名か、親に置く子の鍵のハッシュか
NSEC と NSEC3	名前の順序をそのまま示すか、ハッシュ化して列挙を防ぐか
chroot と TSIG	ファイルシステム上の隔離か、通信の認証か
DANE と DNSSEC	TLSA で証明書を検証するか、レコードに署名して検証可能にするか

この章の必修コマンド一覧

コマンド	何をするか
named	BIND の本体。-t で chroot、-u で権限降格
rndc	稼働中の named へ制御コマンドを送る (reload ・ reconfig ・ retransfer ・ flush ほか)
rndc-confgen	rndc の制御用の鍵と設定を生成する
named-checkconf	named.conf の構文を検査する (-z でゾーンファイルまで)
named-checkzone	ゾーンファイルの構文と整合性を検査する (ゾーン名が先、ファイルが後)
named-compilezone	ゾーンファイルの形式を変換する (-o が必須)

コマンド	何をするか
<code>dig</code>	応答の詳細まで含めた問い合わせ (<code>@</code> <code>-x</code> <code>•</code> <code>+short</code> <code>•</code> <code>+trace</code> <code>•</code> <code>AXFR</code>)
<code>host</code>	簡潔な形式の名前解決
<code>nslookup</code>	対話モードを持つ名前解決
<code>tsig-keygen</code>	TSIG 用の共有鍵を生成する
<code>dnssec-keygen</code>	DNSSEC の鍵ペア (ZSK・KSK) を生成する
<code>dnssec-signzone</code>	生成済みの鍵でゾーンファイルに署名する

試験ポイント

- **aa フラグが無い応答は非権威。** `qr rd ra` だけなら、再帰を要求して受け取ったキャッシュ経由の答えである
- **allow-query は通常の問い合わせ、allow-transfer はゾーン転送。** ゾーン転送を開けるとホスト一覧が丸ごと漏れる
- **forward only は失敗したらそれで終わり。** 自分でルートから解きに行くのは `forward first` (既定)
- **rndc reconfig は新規ゾーンだけ、rndc reload は新規ゾーン+変更されたゾーンファイル。** 既存ゾーンを手編集したなら `reload`
- **セカンダリのデータをマスターから取り直させるのは rndc retransfer。**
`dig AXFR` は表示するだけで更新しない
- **named-checkzone の引数はゾーン名が先、ゾーンファイルが後。** 成功時は `loaded serial` と `OK` を出す (`named-checkconf` は成功時に沈黙する)
- **-f が入力形式、-F が出力形式。** テキストから生形式への変換は `-f text -F raw`

- ゾーンファイルを編集したら必ずシリアル番号を増やす。増やし忘れの症状は「セカンダリだけが古いまま」
- SOA は Refresh (定期確認)・Retry (失敗時の再試行)・Expire (権威を失うまで)・Minimum (否定応答のキャッシュ時間)。Minimum だけリゾルバ側に効く
- MX は数値が小さいほど優先度が高い。値は帯域でも連番の制約でもない
- CNAME を付けた名前に他のレコードは併記できない。MX の指す先に CNAME を使わない
- 逆引きゾーン名はネットワーク部を逆順にして `in-addr.arpa`。
`192.0.2.0/24` なら `2.0.192.in-addr.arpa`
- PTR の値は末尾にドットを付けた完全修飾のホスト名。忘れると原点が二重に付く
- 委任は親に子の NS レコード + (子ゾーン内の名前なら) グルーの A レコード。
- `dig` の問い合わせ先指定は `@` の書式のみ。`-s`・`--server=` は存在しない。逆引きは `-x`
- `chroot` 先には設定・ゾーン・ライブラリ・デバイスファイルを配置し直す。
`-u` による権限降格と併用できる
- TSIG は共有鍵 + HMAC でトランザクションを認証、DNSSEC は公開鍵でレコードに署名。暗号化するのは DoT / DoH
- DNSSEC は `dnssec-keygen` (鍵を作る) → `dnssec-signzone` (署名する) の順。`-f KSK` を付けなければ ZSK
- DS の親ゾーンへの登録は手作業。`dnssec-signzone` は `dsset-` ファイルを作るところまで

- DANE は TLSA レコードで TLS 証明書を検証する仕組みで、DNSSEC による署名が前提。
-

■ サンプルはここまでです

続き(第2章～巻末)は、LPIC-2 202 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

LPIC-2 202試験 学習用テキスト(無料サンプル)

版

2026年10月版(LPI公式 LPIC-2 Exam 202 Objectives Version 4.5 準拠)

発行

IT資格道場(<https://www.shikaku-doujou.com>)

お問い合わせ

info@shikaku-doujou.com

LPI および LPIC は Linux Professional Institute Inc. の商標または登録商標です。Linux は Linus Torvalds 氏の米国およびその他の国における商標または登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式シラバスの内容を基に作成したもので、Linux Professional Institute との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。