Nixpkgs Reference Manual を読む②（Nixpkgs lib）

Mutsuha Asada
@mutsuha_asada

Nix 日本語コミュニティ ゼミ 2026/01/09

前回までのあらすじ

- ソフトウェアを学習する際に最も効率が良いのはマニュアルの初手通読だが、ハードルが高く、行間が広めなのでどうしても手を動かすことで理解しようとしてしまう
- このシリーズ（Nixpkgs Reference Manual を読む）では、マニュアルを通読して、必要に応じて行間を埋めて発表するという形式を取る
- 前回は nixpkgs のプラットフォームに対するサポートの差と、Global configuration について見てきました

目次

1. lib

概要

- nixpkgs が提供するライブラリ（lib）から、面白い関数を 4 個選んで紹介する
- 基本的には少しマニアックなものを拾った
- 知っていると評価が軽くなったり、書く量が減ったりする
- ライブラリ関数はかなりの量があるので、一度はマニュアルを読むことをおすすめしたい

lib とは

- nixpkgs が提供する、Nix 式向けの標準ライブラリ
- 対象領域が広い
  - 文字列
  - リスト
  - attrset（属性セット）
  - モジュールシステム
  - fetcher
- ↔ `builtins` は Nix 言語に組み込まれている

lib.fileset.toSource

- `src = ./.;`のように path をそのまま渡すとディレクトリ全体が store に入って評価やビルドが重くなりやすい
- `lib.fileset` は必要なファイルだけを明示的に集合として扱える
- `toSource` は fileset で指定したファイル群だけを store に入れて `src` にできる
- 用途の例
  - `src` を最小化して評価・ビルド・キャッシュ効率を上げる

lib.fileset.toSource

```
let
  pkgs = import <nixpkgs> {};
  lib = pkgs.lib;
  fs = lib.fileset.unions [
    ./src
    ./README.md
    (lib.fileset.maybeMissing ./LICENSE)
  ];
  src = lib.fileset.toSource { root = ./.; fileset = fs; };
in
pkgs.stdenv.mkDerivation {
  pname = "demo";
  version = "0.1.0";
  inherit src;
}
```

lib.derivations.lazyDerivation

- 非自明な derivation は評価だけでも時間が掛かることがある
- `lib.derivations.lazyDerivation` は、アクセスしそうな属性に限定した薄い属性のみを返す
- 用途の例
  - `passthru` だけ評価したい
  - 巨大な属性セットを舐めたい

lib.derivations.lazyDerivation

```
let
  pkgs = import <nixpkgs> {};
  lib = pkgs.lib;
  heavy = pkgs.runCommand "heavy" {} ''
    mkdir -p $out
    echo ok > $out/ok
  '';
  lazy = lib.derivations.lazyDerivation {
    derivation = heavy;
    passthru = {
      hello = "world";
      tests.smoke = pkgs.runCommand "smoke" {} "mkdir -p $out; echo smoke > $out/x";
    };
  };
in { inherit (lazy) passthru; }
```

lib.fetchers.withNormalizedHash

- fetcher の引数として hash、sha256、sha512 を許す API を作りたいことがある
- ただし内部では outputHash、outputHashAlgo を使いたい
  - mkDerivation を呼び出すため
- withNormalizedHash は外側の入力は柔軟に受け入れて、内側を正規化する薄いラッパー
- 用途の例
  - fetcher を自作する際に hash と sha256 の両方に対応したい

lib.fetchers.withNormalizedHash

```
let
  pkgs = import <nixpkgs> {};
  lib = pkgs.lib;
  myFetch =
    lib.fetchers.withNormalizedHash { hashTypes = [ "sha256" "sha512" ]; } (
      { url, outputHash, outputHashAlgo, ... }:
      pkgs.fetchurl {
        inherit url outputHash outputHashAlgo;
      }
    );
in
myFetch {
  url = "https://example.com/archive.tar.gz";
  sha256 = "sha256-AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=";
}
```

lib.attrsets.updateManyAttrsByPath

- ネストした属性セットの特定のキーだけをまとめて更新したいときに便利
- `{ path = ["a" "b"]; update = old: ...; }`の形で更新内容を記述する
- 深いパスが先に適用される
- 用途の例
  - 設定の部分的な大量の`//`と `recursiveUpdate` を減らせる

lib.attrsets.updateManyAttrsByPath

```
let
  pkgs = import <nixpkgs> {};
  lib = pkgs.lib;

  x = {
    a = { b = 1; c = 2; };
    d = 10;
  };

  y = lib.attrsets.updateManyAttrsByPath [
    { path = [ "a" "b" ]; update = old: old + 100; }
    { path = [ "a" "c" ]; update = old: old * 3; }
    { path = [ "e" ];     update = old: 999; } # 属性`e`が無い場合は`old`を参照するとエラー
  ] x;
in
y
```

2. まとめ

まとめ

- `lib` は nixpkgs が巨大な集合を扱うために積み上げてきたライブラリ
- 今日紹介したのは 4 つの関数
  - `lib.fileset.toSource`
  - `lib.derivations.lazyDerivation`
  - `lib.fetchers.withNormalizedHash`
  - `lib.attrsets.updateManyAttrsByPath`
- Nix の性質に合わせて設計されたライブラリを使うことで、nixpkgs らしい書き方に自然と近づく
- 次回は、`lib` の中でも特に使用頻度が高い関数群をもう少し体系的に見ていく