

# Deno の話

**Deno 使ってますか？** 🙋🙋

# 話す人

日野澤 歓也 twitter @kt3k

- GREE (2012 - 2013)
- Recruit (2015 - 2019)
- Deno Land (2021 -)

2018年から Deno にコントリビュートを開始。2020年作者に誘われ Deno

Land に転職。現在はフルタイムで Deno と Deno Deploy を開発中。




# Deno のロードマップ

## **Deno のこれまでのロードマップ**

- **Web 互換性**
- **TypeScript サポート**
- **ESM サポート**
- **V8 サンドボックスセキュリティ**
- **Node を上回るHTTPパフォーマンス**

## Deno のこれまでのロードマップ

- Web 互換性 ▲
- TypeScript サポート 
- ESM サポート 
- V8 サンドボックスセキュリティ 
- Node を上回るHTTPパフォーマンス 

**v1.0 時点**

## Deno のこれまでのロードマップ

- Web 互換性 
- TypeScript サポート 
- ESM サポート 
- V8 サンドボックスセキュリティ 
- Node を上回るHTTPパフォーマンス ▲

**v1.16 (最新)** では大部分達成

# Deno のこれからのロードマップ



## **Deno のこれからのロードマップ**

- **さらなるHTTPパフォーマンス向上**
- **Node.js 互換性 (後述)**

**Q. Node製のライブラリはDenoで使えるか**

**Q. Node製のライブラリはDenoで使えるか**

**A. 主に2つの方法がある (どちらも完璧ではない)**

- **npm モジュールを Deno 用に変換する CDN を介して使う方法 (非公式)**
- **Deno の Node.js 互換モードを使う(2022 Q2 予定)方法 (公式)**

# 1) npm モジュールを Deno 用に変換する CDN を介して使う (非公式)

- **esm.sh**

`https://esm.sh/<package名>`

- **skypack**

`https://cdn.skypack.dev/<package名>`

**React などはこの手法で動く**

```
import React from "https://esm.sh/react";  
import ReactDOM from "https://esm.sh/react-dom";
```

## 2) Deno の Node.js 互換モードを使う

- 現在、Deno コアチームにて鋭意開発中
- 2022 Q2 の v2 リリースである程度使える状態にする事が目標

```
npm install express
deno run --compat --unstable -A server.js
```

```
const express = require('express')
const app = express()
app.get('/', (req, res) => {
  res.send('Hello World!')
})
console.log('Listening on localhost:3000');
app.listen(3000)
```

**Node.js 互換モードについては [v1.5 リリース記事](#) に  
概要の記述があります。**

**Q. 権限管理のベストプラクティスは？**

**Q. 権限管理のベストプラクティスは?**

**A. 必要最小限のものを渡す**

**例. リンター**

```
deno run --allow-read linter.ts
```

**例. フォーマッター**

```
deno run --allow-read --allow-write formatter.ts
```



**Q. 権限管理のベストプラクティスは?**

**A. ネットワークパーミッションを渡す場合はホスト名を指定する**

**例. ポート8080で動く、Postgres と通信するサーバー**

```
deno run \  
  --allow-net=localhost:8080,host-to.postgres:5432 \  
  server.ts
```

**Q. 権限管理のベストプラクティスは?**

**A. 環境変数、プロセス実行は許可範囲を出来るだけ絞る**

```
deno run \  
  --allow-env=AWS_ACCESS_KEY_ID,AWS_SECRET_ACCESS_KEY \  
  do_something_in_aws.ts
```

```
deno run --allow-run=git do_something_with_git.ts
```

## Q. 権限管理のベストプラクティスは?

**Note:** `--allow-run` (引数なし) `--allow-run=deno` は `--allow-all` (すべて許可) と同じとなるため注意

以下のコードで権限の昇格が可能

```
Deno.run({  
  cmd: ["deno", "run", "--allow-all", import.meta.url]  
});
```

# Webサーバ用のライブラリ比較

## Webサーバ用のライブラリ比較

- Webフレームワークが20個ぐらいある。
- 以下おすすめを紹介

# 1) oak

- 一番人気
- koa にインスパイアされたデザイン

```
import { Application, Router }  
  from "https://deno.land/x/oak/mod.ts";  
  
const router = new Router()  
  .get("/", (context) => {  
    context.response.body = "Hello world!";  
  })  
  .get("/book/:id", (context) => {  
    context.response.body = books.get(context.params.id)  
  });  
  
const app = new Application();  
app.use(router.routes());  
await app.listen({ port: 8000 });
```

## 2) `std/http` + [URLPattern](#) API

- `std/http` は Deno 標準モジュールが提供する http サーバー機能

```
import { serve }  
  from "https://deno.land/std@0.114.0/http/server.ts";  
  
const topPage = new URLPattern({ pathname: "/" });  
const myPage = new URLPattern({ pathname: "/me" });  
  
console.log("Listening on http://localhost:8000");  
await serve((req) => {  
  if (topPage.test(req.url))  
    return new Response("Top page");  
  if (myPage.test(req.url))  
    return new Response("My page");  
  return new Response("Not Found", { status: 404 });  
});
```

### 3) sift

- **JSX 対応などが便利、ミニマルな API**

```
/** @jsx h */
import { h, jsx, serve }
  from "https://deno.land/x/sift@0.4.2/mod.ts";

const App =
  () => <div><h1>Hello world!</h1></div>;

const NotFound =
  () => <div><h1>Page not found</h1></div>;

serve({
  "/" : () => jsx(<App />),
  404 : () => jsx(<NotFound />, { status: 404 }),
});
```



**Q. When will deno be production ready**

## Q. When will deno be production ready

- It depends on what you need in production.
- ランタイムの安定性という意味では既に **production ready** と言って良い段階
- API も 1.0 以降は **semver** を遵守しているため、バージョンアップで急に壊れたりはない
- **npm** モジュールが使えなければ困るという事であれば **2022 Q2 (目標)** の **Node.js 互換モード** を一旦待ってください

**Q. deno が TS を捨てたときの裏話? があれば聞きたい**

**[https://docs.google.com/document/d/1\\_WvwHI7BXUIpru=AAABcrrKL5k\\*nQ4LS569NsRRAce2BVanXw](https://docs.google.com/document/d/1_WvwHI7BXUIpru=AAABcrrKL5k*nQ4LS569NsRRAce2BVanXw)**

## deno が TS を捨てた話

- 内部 API 実装で Deno は最初は TS を使っていたが、それを辞めて素の JS を使うようになったという話

## deno が TS を捨てた話

- 内部 API の TS は Deno のユーザーが使う TS とは全く別物で独自の専用プログラムでバンドルしていた
  - そもそもメンテできる人が限られていた
- dts 定義なども独自ツールで生成していたが、そのツールのバグで特定の型の API が定義出来ない不都合などがあった
  - メンテできる人もほぼ1人だった

=> 辛さが限界に達し、TS をやめた方が良いという判断になった

## Q. ORM比較

## ORM 比較


- 現状だと **denodb** しか選択肢がなさそう。  
参考: [Deno で試すデータベースアクセス](#)
- [prisma](#) は未対応
- マイグレーションは [Nessie](#) がそれなりに動いてる模様

(Deno コアチームは ORM に対する意識がおそらくかなり低い)

宣伝



# Deno Deploy

 deploy

CLI   Blog   Documentation   Sign in   [Sign up](#)

## Globally Distributed JavaScript

- ⚡ Instant deployments
- 🌐 Located in 28 regions worldwide
- ⚙️ Zero config, zero maintenance
- ✓ TypeScript, WASM, ES Modules

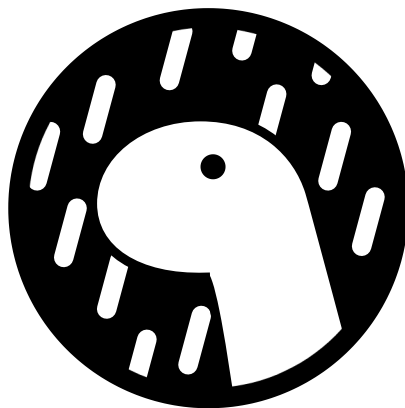
[Sign up](#)

```
1 import { serve } from "https://deno.land/std@0.114.0/http/server.ts"
2 import { h, ssr, tw } from "https://crux.land/nanosr@0.0.1"
3
4 const Hello = (props) => (
5   <div class={tw`bg-white flex h-screen`}>
6     <h1 class={tw`text-5xl text-gray-600 m-auto mt-20`}>
7       Hello {props.name}!
8     </h1>
9   </div>
10 );
11
12 console.log("Listening on http://localhost:8000");
13 await serve((req) => {
14   const url = new URL(req.url);
15   const name = url.searchParams.get("name") ?? "world";
16   return ssr(() => <Hello name={name} />);
17 });
```

[Deploy](#)   [Documentation](#)

# Deno Deploy

- Deno と互換な Serverless at Edge サービス
- Deno の [サブセットとなる API](#) をサポート
- 世界 [28 のデータセンター](#) に自動的にデプロイ
- 非常に高速なコールドスタートアップタイム (\*)
- DB 接続も可能 [参考](#)
- 前述の `oak` , `sift` , `std/http` などのフレームワークもそのまま利用可能



**End**