APIハンズオン

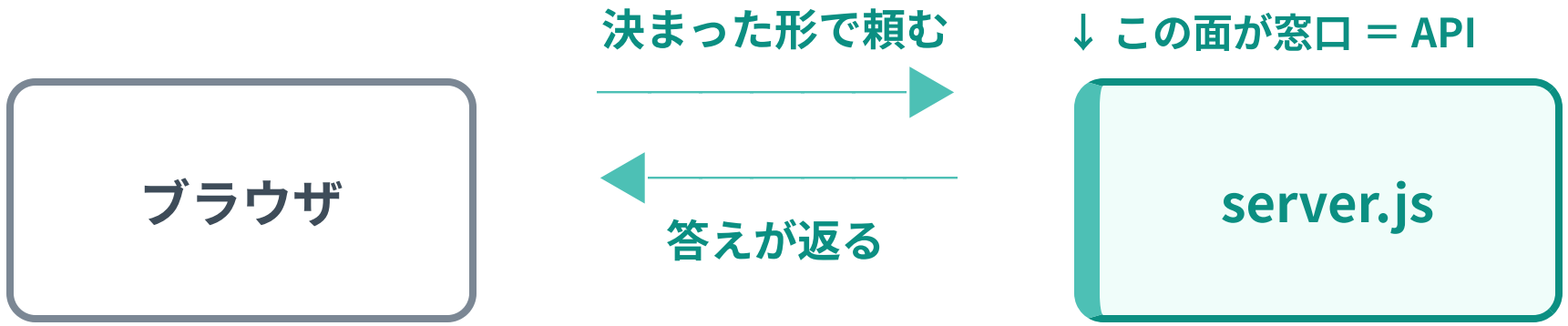
1. APIとは何か

名前がそのまま意味

Application	アプリケーションを
Programming	プログラムから使うための
Interface	**窓口**

アプリをプログラムから使うための窓口

インターフェース ＝ 窓口



inter（間） + face（面） ＝ **間にある面** 厚みのない1枚の面
窓口は**サーバー側の表面** 銀行の窓口が銀行の一部なのと同じ

窓口があると うれしいことが2つ

利点1 中身を知らなくても使える

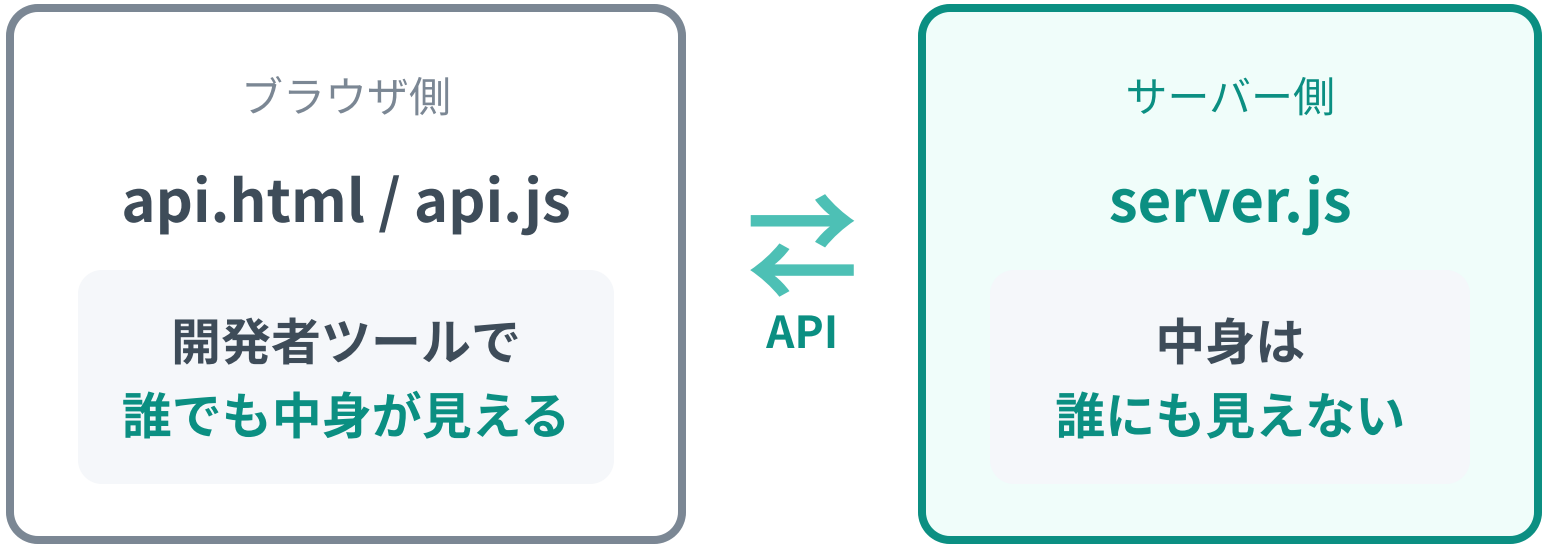
利点2 見せたくないものを隠せる

利点1 中身を知らなくても使える

	裏側でやっていること	自分が書くこと
天気	観測データを集めて スパコンで計算	`fetch("/weather")`
地図	膨大な画像から その地点を切り出す	`fetch("/map")`
翻訳	大量の文章で学習したモデルで推論	`fetch("/translate")`

左は**知らなくていい** 書くのは右の**1行**だけ
自分で気象観測をしなくても 天気を出すアプリがつくれる

利点2 見せたくないものを隠せる



パスワードの照合 APIキー データベースの情報は **全部この右側に置く**
この見えない側を**バックエンド**と呼ぶ

窓口のルールは この4つ

	これがルール	今日つくるもの
①	どのURLか	`/auth`
②	どのメソッドか	**`POST`**
③	何を渡すか	`password`
④	何が返るか	`{ ok, message }`

この4つが決まっていれば **別々の人がつくっても繋がる**
②のGETとPOSTが 前の章でやったやつ

4つを組み合わせると コードになる

```
// server.js
if (req.method === "POST" && pathname === "/auth") {
//      ② メソッド                        ① URL

  const body = await req.json();
//      ③ 送られたもの を受け取る

  return Response.json({ ok: true, message: "ログインできました" });
//                      ④ 返すもの
}
```

今日書くのは **これだけ**
API を増やすときも ①②③④を決めるだけ

2. server.jsは
全部の入口

これがテンプレートの最初の形

```
import { serveDir } from "jsr:@std/http/file-server";

Deno.serve((req) => {                          // 通信は全部ここに来る
  const pathname = new URL(req.url).pathname;
  console.log(pathname);                       // 叩かれたパスを表示

  if (req.method === "GET" && pathname === "/welcome-message") {
    return new Response("jigインターンへようこそ！");
  }

  return serveDir(req, { fsRoot: "public", ... });
});
```

テンプレートから始まったときの `server.js`

書き換えたのは中身だけ **形はこのまま残っている**

上から順に「自分の担当か」を見ている

通信が来た

1. GETで `/welcome-message` か？ → **バックエンドの情報** を返す
2. **どれでもない → `serveDir` が `public` の中のファイル を返す**

一番下の `serveDir` は「どれにも当たらなかったとき」の受け皿

`public` の中のファイルも server.js が返している

画面をつくる部品も データも **同じ入口から 同じ形で返る**
だから 1ファイルで両方できている

1つずつ 4回に分けて返る

叩かれたURL	上から順に見た結果	返ってきたもの
`/`	どれにも当たらない → `serveDir`	`index.html`
`/styles.css`	どれにも当たらない → `serveDir`	CSS
`/index.js`	どれにも当たらない → `serveDir`	JavaScript
`/welcome-message`	**1番目のifに当たった**	「ようこそ」の文字

3回は **`public` の中のファイル** 1回は **バックエンドの情報**
どちらも同じ `server.js` が 同じ形で返している

自分のサイトで見てみよう

1. **開発者ツール**（F12）→ **Network** タブ
2. **ページを再読み込みすると 4行ならぶ**

Name	Status	Type	Initiator	Size	Time
deno-app-21-v38jsdz534q...	200	docum...	Other	0.6 kB	189 ms
styles.css	200	styles...	(index):8	(memo...	0 ms
index.js	200	script	(index):9	(memo...	0 ms
welcome-message	200	fetch	index.js:1	0.1 kB	191 ms

`Initiator` を見ると **誰が呼んだか**が分かる

`styles.css` と `index.js` は `(index)` から つまり**HTMLが呼んでいる**

3. APIハンズオン

デプロイしたアプリに 足していく

1. クローンしたフォルダを開く 例: deno-app
2. `server.js` に窓口を**1つ**足す
3. **直すたびに push する** 今日は自分のPCでは動かさない デプロイ先のURLで確かめる
4. デプロイ先のURLに `/api.html` を付けると **スマホからも使える**

作業するのは**自分のリポジトリ**
教材リポジトリ（`intern-dev-tutorial`）**ではない**

進め方

1. **サーバー側**に書く server.js
2. **ブラウザ側**を新しくつくる api.html / api.js
3. **push する**
4. `/api.html` を開いて ボタンで確かめる

保存しただけでは **反映されない** 動いているのは**GitHubに置いたコード**

今日さわるファイルは3つ

ファイル	今日の扱い
`server.js`	`return serveDir(` の**上に足す**
`public/api.html`	**新しくつくる** 今日の画面
`public/api.js`	**新しくつくる** 今日書くJSは全部ここ

もとの `index.html` は **触らない 別のページ**をつくる

`serveDir` より下に書くと**たどりつかない**ので 必ず**その上**へ

READMEへ！