

# Web認証

## 基礎講座

React + Hono デモアプリを使う

## 今日覚えること

1. 認証って何？ 身近な例で理解
2. なぜ必要？ Webアプリの課題
3. どうやって実現？ 4つの方法

# 認証って何？

## 身近な例で考えてみよう

**認証** = 本人確認

- スマホの指紋認証
- 銀行ATMの暗証番号
- 会社の入館カード

**認可** = 権限確認

- 管理者だけがアクセスできるページ
- 自分の口座しか見れない
- 部長しか承認できない機能

# なぜWebで認証が必要？

問題：誰でもアクセスできてしまう

```
http://example.com/mypage
```

↓

誰でも他人のマイページが見れる！

解決：ログインした人だけ

ログイン画面 → 本人確認 → マイページ表示

つまり：「この人は確かに〇〇さんです」を証明する仕組み

## ステートレス と ステートフル

### ステートフル（状態を覚える）

田中：「もしもし」  
店員：「田中さんですね」  
田中：「注文したいです」  
店員：「田中さんのご注文ですね」

店員は「田中さんと話している」ことを覚えている

# ステートレス と ステートフル

## HTTPはステートレス設計

### 1回目のリクエスト

あなた：「ログインしました！」  
サーバー：「OKです。」

a few moments later...

### 2回目のリクエスト

あなた：「マイページ見せて」  
サーバー：「あなた誰でしたっけ？」

**サーバーは1回目と2回目が同じ人かわからない！**

# なぜステートレスなのにセッションが必要になったのか？

## Webの歴史：最初は「文書の共有」だった

### 1990年代初期のWeb

大学の論文 → HTML → ブラウザで見る  
研究資料 → HTML → ブラウザで見る

- 単なる 文書の閲覧
- 「前に何を見たか」を覚える必要がない
- ステートレス = 理にかなっていた

# Webアプリの進化：「交流」が始まった

## 1990年代後半：掲示板、メール

### 問題発生

掲示板：「誰が書き込んだか分からない」

メール：「ログインしても次のページで忘れられる」

## 2000年代：ショッピングサイト

### さらに複雑に

カートに商品追加 → 「誰のカート？」

決済画面へ → 「また1から入力？」

### 結果：ステートレス設計では限界



## 解決方法：覚えておいてもらう

### 24 お店の例

1. 入店時：「整理券22番です」
2. 買い物中：「22番の方ですね」
3. 退店時：「22番の方、ありがとうございました」

### Webでも同じ

1. ログイン時：「セッションID ABC123」
2. ページ閲覧時：「ABC123の方ですね」
3. ログアウト時：「ABC123さん、お疲れさまでした」



## 2つの考え方



### セッション方式（サーバーに状態を保存）

- クライアント: 整理券「ABC123」を渡す
- サーバー: 「ABC123 = 田中さん」をメモリに保存
- サーバー: 「ABC123? 田中さんですね」



### JWT方式（すべて自己完結）

- クライアント: 身分証明書「私は田中、有効期限〇〇」を渡す
- サーバー: 何も保存しない
- サーバー: 「証明書確認しました、田中さんですね」

両方ともクライアントが情報を渡すが、サーバーの動作が違う

# 実際の認証方法を見てみよう

## 4つの主要な方法

1. 基本認証 - 一番シンプル
2. セッション認証 -  お店型の実装
3. JWT認証 -  スマホ型の実装

それぞれメリット・デメリットがあります  
実際のコードも見ながら理解していきましょう


# 方法1：基本認証

## 一番シンプルな方法


ユーザー名: tanaka

パスワード: password123

## 良いところ・悪いところ

 簡単に実装できる

 パスワードが丸見え（Base64のみ）

 毎回入力が必要

## 基本認証の問題を解決するには？

### 基本認証の問題点

- 毎回パスワード入力（面倒）
- パスワードが見えてしまう（危険）
- ログアウト機能がない

### 解決策：「整理券」方式

- 最初だけパスワード → 整理券もらう
- 以降は整理券を見せるだけ
- 整理券を捨てればログアウト




これが「セッション認証」の考え方

## 方法2：セッション認証

### サーバーが状態を覚える方式

1. ログイン成功 → セッションID「ABC123」発行
2. サーバー: メモリに「ABC123 = 田中さん」を保存
3. ブラウザ: 「ABC123です」
4. サーバー: 「ABC123...田中さんですね」

## 良いところ・悪いところ

-  サーバーが全部管理（安全）
-  すぐにログアウトできる（メモリから削除）
-  ユーザーが増えると重くなる（メモリ使用量増加）

## セッション認証の限界

### 大きなサービスでの問題

- ユーザー100万人 → サーバーに100万個の整理券保存
- サーバーが重くなる、メモリ不足
- 複数サーバー → 整理券の共有が大変

### 別のアプローチ：「身分証明書」方式

- 整理券ではなく、詳細な身分証を渡す
- 身分証に「名前、権限、有効期限」を書く
- サーバーは覚えなくていい → 軽い！




## これが「JWT認証」の考え方

## 方法3：JWT認証

### サーバーが状態を持たない方式

1. ログイン成功 → JWT 「私は田中、権限:一般、期限:2025/01/01」 発行
2. サーバー: 何も保存しない
3. ブラウザ: 「私は田中、権限:一般、期限:2025/01/01」
4. サーバー: 「署名確認...田中さんですね」

### 良いところ・悪いところ

-  サーバーが軽い（何も保存しない）
-  大きなシステムに向いている
-  一度発行すると止められない（サーバーが覚えていないから）



# ⚠ JWT認証の落とし穴

## ログアウトできない？

### 普通の認証（セッション）

ログアウト → サーバー「はい、削除しました」  
→ 本当にログアウト完了

### JWT認証

ログアウト → ブラウザから削除  
→ でもサーバーは「まだ有効」と思ってる  
→ 見た目だけのログアウト

**重要：盗まれたトークンは期限まで使われる可能性**

# JWT認証の問題をどう解決する？

## 現実的な問題

- JWTは一度発行すると止められない
- セキュリティ事故が起きても手遅れ
- 自分で認証システムを作るのは大変

## プロが作った解決策

- **AWS Cognito** = Amazonが作った認証サービス
- JWTの良いところ + ログアウト機能
- セキュリティもプロ仕様

「車輪の再発明」をしなくていい

## 方法4：AWS Cognito

### Amazon が提供するサービス

- ユーザー登録・ログイン機能
- SMS認証、顔認証なども簡単
- トークンを無効化できる（JWTの問題を解決）

### 良いところ

- 😊 自分で作らなくていい
- 😊 セキュリティ機能が充実
- 😊 ユーザーが増えても大丈夫