「え？！それ今ではHTMLだけでできるの！？」驚きの進化を遂げたモダンHTML

自己紹介

- 名前：西 悠太
- 所属：株式会社ダイニー
- TypeScriptが好きです
- 自称フロントエンドエンジニア
  (肩書はPlatform Engineer)

今日のゴール

「へー！今のHTMLってそんなことできるんだー」

と思って帰ってもらう

HTML Living Standardの時代へ

2019年5月、W3CとWHATWGはHTMLとDOM標準の開発を WHATWGが主導することで合意しました

これにより、HTMLは「HTML5」のようなバージョン番号を持つ仕様から、 継続的に更新される「HTML Living Standard」へと移行しました

HTMLの進化の方向性

この変化により、以下のような傾向が明確に現れています

宣言的UI構築への移行 JavaScript実装から、HTML属性による宣言的な記述へ

ブラウザネイティブ最適化 パフォーマンスやアクセシビリティをブラウザレベルで最適化

開発者体験の向上 より直感的で保守しやすいマークアップの実現

今日紹介する機能

1. Popover API (Baseline 2024)
2. Dialog要素 (Baseline 2022)
3. details要素のname属性 (Baseline 2024)
4. inert属性 (Baseline 2023)
5. search要素 (Baseline 2023)
6. loading属性 (Baseline 2020)
7. fetchpriority属性 (Baseline 2023)
8. blocking属性 (Baseline 2024)
9. inputmode属性 (Baseline 2018)
10. enterkeyhint属性 (Baseline 2021)
11. rel属性のSEO対応値 (2019)

1. Popover API (Baseline 2024)

ネイティブなポップオーバー機能

ポップアップ実装で困ること

- z-indexの管理が大変
- 外側クリックで閉じる処理
- ESCキーで閉じる処理
- フォーカス管理

結局ライブラリに頼ることに...

今はHTMLだけで解決

popovertarget属性 + popover属性を指定するだけ

→ z-index管理、外側クリック/ESCキーで閉じる、フォーカス管理を自動処理

```
<button popovertarget="menu">メニューを開く</button>
<div popover="auto" id="menu">ポップアップの内容</div>
```

なぜz-index問題が解決するの？

トップレイヤーという新しい描画層

- DOM階層から完全に独立
- z-indexの制約を受けない
- `overflow:hidden` で切れない

従来のz-index地獄から解放

従来の実装 vs Popover API

従来の実装
管理が複雑

```
<button onclick="togglePopup()">
  メニュー
</button>
<div id="popup" class="popup hidden">
  ...
</div>
<script>
function togglePopup() {
  // 色々な処理が必要
}
</script>
```

Popover API
トップレイヤーで独立 + 自動イベント管理

```
<button popovertarget="menu">
  メニュー
</button>
<div popover id="menu">
  ...
</div>
```

popovertargetaction属性

同じポップオーバーに対して、ボタンごとに異なる操作を指定できます

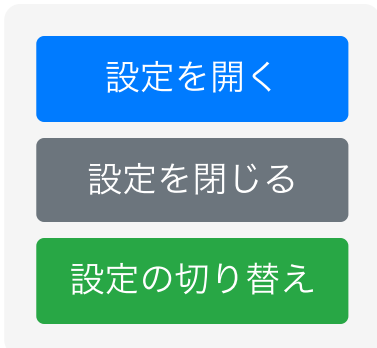
値	動作	使用場面
show	ポップオーバーを開く	「開く」ボタン
hide	ポップオーバーを閉じる	「閉じる」ボタン
toggle	開閉を切り替える（デフォルト）	トグルボタン

popovertarget属性による宣言的関係性

属性で関係性を宣言 → 自動的に紐づけ

```
<!-- 複数のボタンで同じポップオーバーを制御 -->
<button popovertarget="settings">設定を開く</button>
<button popovertarget="settings" popovertargetaction="hide">設定を閉じる</button>
<button popovertarget="settings" popovertargetaction="toggle">設定の切り替え</button>
<div popover="auto" id="settings">
  <h2>設定</h2>
  <p>ここに設定内容が入ります。</p>
</div>
```

適切なイベントハンドリング + アクセシビリティ属性の自動設定


設定を開く
設定を閉じる
設定の切り替え

2. Dialog要素 (Baseline 2022)

ネイティブなモーダルダイアログ

モーダルダイアログ実装で困ること

- フォーカストラップの実装
- ESCキーで閉じる処理
- 背景の無効化
- アクセシビリティ対応

自前で全部実装するのは大変...

今はDialog要素で解決

Dialog要素で以下を自動処理：

フォーカストラップ、ESCキーで閉じる、背景の無効化、アクセシビリティ対応

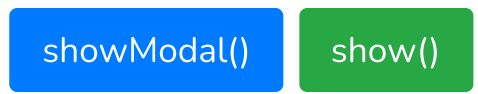
モーダルダイアログの実装

```
<dialog id="my-dialog">
  <h2>ダイアログタイトル</h2>
  <p>ダイアログの内容がここに入ります。</p>
  <button id="close-dialog">閉じる</button>
</dialog>

<button id="open-dialog">ダイアログを開く</button>

<script>
  const dialog = document.getElementById('my-dialog');
  document.getElementById('open-dialog').addEventListener('click', () => {
    dialog.showModal(); // モーダルダイアログとして表示
  });
  document.getElementById('close-dialog').addEventListener('click', () => {
    dialog.close();
  });
</script>
```

showModal()とshow()の違い

showModal() → モーダル表示 + `::backdrop` で背景を覆う + フォーカストラップ自動実装

show() → 非モーダル表示、背景の要素も操作可能


showModal()
show()

form要素との統合

`method="dialog"` 指定 → 送信時にダイアログを自動で閉じる + ボタンのvalueを `dialog.returnValue` に設定

```
<dialog id="confirm-dialog">
  <form method="dialog">
    <h2>削除の確認</h2>
    <p>この操作は取り消せません。</p>
    <button value="cancel">キャンセル</button>
    <button value="confirm">削除</button>
  </form>
</dialog>
```

DialogとPopoverの使い分け

特性	Dialog	Popover
フォーカストラップ	✅ 自動（モーダル時）	❌ なし
背景の無効化	✅ ::backdrop擬似要素	❌ 手動実装必要
ESCキーで閉じる	✅ 自動	✅ 自動
form統合	✅ method="dialog"	❌ なし
用途	確認、入力が必要	情報表示、メニュー

3. details要素のname属性 (Baseline 2024)

ネイティブなアコーディオン

アコーディオン実装で困ること

一度に1つだけ開くアコーディオン

従来のdetails要素は独立動作 → JavaScriptで状態管理が必要

複数が同時に開いてしまう

今はname属性で解決

同じname値を持つdetails要素 → 自動で排他制御、JavaScript不要

アコーディオンの実装例

```
<!-- FAQセクション：一度に1つの質問だけ開く -->
<details name="faq">
  <summary>返品は可能ですか？</summary>
  <p>商品到着後14日以内であれば返品可能です。</p>
</details>

<details name="faq">
  <summary>送料はいくらですか？</summary>
  <p>5,000円以上のご購入で送料無料です。</p>
</details>

<details name="faq">
  <summary>支払い方法は？</summary>
  <p>クレジットカード、銀行振込、代金引換をご利用いただけます。</p>
</details>
```

JavaScriptゼロ行

▶ 返品は可能ですか？

▶ 送料はいくらですか？

▶ 支払い方法は？

4. inert属性 (Baseline 2023)

包括的な要素の無効化

非表示スライド内のリンクを無効化したい時

カルーセルUIで前後のスライドも操作可能になってしまう

従来の`disabled`属性はフォーム要素のみ

→ inert属性でリンクやコンテンツも含めて無効化可能

無効化される範囲：

- フォーカス
- クリック/タップ
- アクセシビリティツリー

各無効化手法の比較

特性	`aria-hidden`	`inert`	`disabled`
対象範囲	アクセシビリティツリーのみ	視覚・操作・アクセシビリティ	フォーム要素のみ
マウス操作	操作可能	操作不可	操作不可（フォーム要素）
フォーカス	可能	不可	不可（フォーム要素）
適用可能要素	すべて	すべて	フォーム要素のみ

従来の`disabled`属性はフォーム要素にしか使えませんでしたが、`inert`属性はあらゆるHTML要素に適用できるので、より柔軟に無効化できます

モーダル表示時の活用例

```
// モーダルを開く際にメインコンテンツを無効化
document.getElementById('main-content').inert = true;
```

inert属性をtrueに設定するだけで、 その要素とすべての子要素がフォーカス不可・クリック不可・アクセシビリティツリーから除外されます

5. search要素 (Baseline 2023)

検索UIの標準化

検索UIのマークアップ

従来：`role="search"`でアクセシビリティ確保

今は：`<search>`要素で包むだけ → スクリーンリーダーが自動で検索機能を認識

使用例

```
<!-- サイト内検索 -->
<search>
  <form action="/search" method="get">
    <label for="site-search">サイト内を検索:</label>
    <input type="search" id="site-search" name="q" required>
    <button type="submit">検索</button>
  </form>
</search>
```

アクセシビリティツリーで `search` ランドマークとして認識 → スクリーンリーダーユーザーが検索機能を素早く発見

6. loading属性 (Baseline 2020)

リソース読み込み制御

画像の遅延読み込み

従来：Intersection Observer APIでゴリゴリ実装

今は：`loading="lazy"`を付けるだけ → 最適なタイミングで自動読み込み

```
<!-- ファーストビューの重要な画像 -->
<img src="hero-image.jpg" loading="eager" alt="メインビジュアル">

<!-- スクロール後に表示される画像 -->
<img src="product-1.jpg" loading="lazy" alt="商品画像1">
<img src="product-2.jpg" loading="lazy" alt="商品画像2">

<!-- 外部コンテンツの遅延読み込み -->
<iframe src="video-player.html" loading="lazy" title="動画プレイヤー"></iframe>
```

使い分け

値	動作	使用場面
lazy	ビューポート接近時に読み込み	フォールド下の画像
eager	即座に読み込み	ファーストビューの画像

7. fetchpriority属性 (Baseline 2023)

リソース優先度制御

LCPが改善しない時

LCP画像を優先的に読み込みたいのに、ブラウザの優先度判断が意図と違う...

→ **今はfetchpriority属性でLCP改善が可能**

- ヒーロー画像 → `high`
- 分析スクリプト → `low`

特にLCP改善に効果的です

使用例

```
<!-- LCP要素となるヒーロー画像を最優先 -->
<img src="hero-banner.jpg" fetchpriority="high" alt="メインビジュアル">

<!-- 重要なスタイルシート -->
<link rel="stylesheet" href="critical.css" fetchpriority="high">

<!-- 優先度の低い装飾画像 -->
<img src="decoration.png" fetchpriority="low" loading="lazy" alt="装飾">

<!-- 分析スクリプトは低優先度 -->
<script src="analytics.js" fetchpriority="low" async></script>
```

優先度の使い分け

優先度	対象リソース	効果
high	LCP画像、クリティカルCSS、重要なフォント	より早く読み込まれる
low	装飾画像、分析スクリプト、非表示コンテンツ	他のリソースを優先
auto	その他の一般的なリソース	ブラウザのデフォルト動作

8. blocking属性 (Baseline 2024)

レンダリング制御

フォント読み込みでチラつく問題

フォントが読み込まれる前にテキストが表示されてチラつく

従来は暗黙的にレンダリングをブロック、明示的に制御不可

→ 今はblocking属性でチラつきを防止可能

必要なリソースが読み込まれてから表示

使用例

```
<!-- 通常のスクリプトはレンダリングを停止 -->
<script src="library.js"></script>

<!-- deferはDOM構築完了後に実行 -->
<script src="framework.js" defer></script>

<!-- preload + blocking="render"でレンダリングを停止 -->
<link rel="preload"
    href="critical-font.woff2"
    as="font"
    blocking="render"
    crossorigin>
```

初期表示に必要不可欠なリソース vs 後から適用可能なリソースを明確に区別

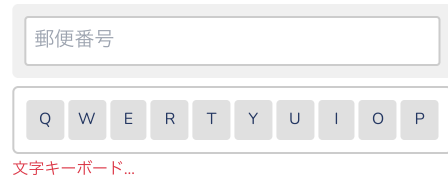
9. inputmode属性 (Baseline 2018)

仮想キーボード最適化

仮想キーボードの最適化

郵便番号入力で数値専用キーボードを出したいのに文字キーボードが表示されてしまう

→ inputmode属性で数値専用キーボードを表示可能




```
<!-- 数値専用キーボード -->
<input type="text" inputmode="numeric" pattern="[0-9]*"
      placeholder="郵便番号（ハイフンなし）">

<!-- 電話番号用キーボード -->
<input type="tel" inputmode="tel" placeholder="090-1234-5678">

<!-- URL入力用キーボード -->
<input type="url" inputmode="url" placeholder="https://example.com">

<!-- メールアドレス用キーボード -->
<input type="email" inputmode="email" placeholder="user@example.com">
```

inputmodeの種類

inputmode	表示されるキーボード	適した用途
numeric	0-9の数字のみ	認証コード、郵便番号
tel	電話番号用（+や-を含む）	電話番号
decimal	数字と小数点	価格、数量
email	@や.comキーを含む	メールアドレス
url	/や.comキーを含む	URL入力
search	検索ボタン付き	検索フィールド

10. enterkeyhint属性 (Baseline 2021)

Enterキー表示の最適化

Enterキー表示の最適化

検索フィールドなのにEnterキーが「改行」と表示されてしまう

→ enterkeyhint属性でユーザーに次のアクションを直感的に示せる


```
<!-- 検索フィールド -->
<input type="search" enterkeyhint="search" placeholder="サイト内を検索">

<!-- 複数ステップフォーム -->
<input type="text" enterkeyhint="next" placeholder="お名前">

<!-- フォームの最終項目 -->
<textarea enterkeyhint="done" placeholder="コメント"></textarea>

<!-- チャットアプリ -->
<input type="text" enterkeyhint="send" placeholder="メッセージを入力">
```

enterkeyhintの種類

値	Enterキー表示	使用場面
search	検索	検索フィールド
next	次へ	フォームの途中フィールド
done	完了	フォームの最終フィールド
go	移動	URL入力フィールド
send	送信	メッセージ入力フィールド

11. rel属性のSEO対応値 (2019)

リンク性質の明確化

広告リンクとユーザー投稿リンクの区別

全部`nofollow`で一括りにしてませんか？

2019年9月にGoogleが発表した新しい`rel`属性値でリンクの性質を詳細に伝達

- `sponsored`：広告やアフィリエイトリンク
- `ugc`：ユーザー生成コンテンツ内のリンク

検索エンジンがリンクの文脈を正確に理解 → PageRankの評価を適切に調整

使用例

```
<!-- 有料広告やスポンサーリンク -->
<a href="https://sponsor.com" rel="sponsored">スポンサーリンク</a>

<!-- ユーザー生成コンテンツ内のリンク -->
<a href="https://user-content.com" rel="ugc">ユーザー投稿のリンク</a>

<!-- 複数の値を組み合わせる -->
<a href="https://untrusted-site.com" rel="nofollow sponsored">
  有料の外部リンク
</a>
```

rel属性の値

値	説明	使用場面
sponsored	広告、スポンサーシップ、金銭的対価のあるリンク	アフィリエイトリンク、記事広告など
ugc	ユーザー生成コンテンツ内のリンク	ブログコメント欄、フォーラム投稿内のリンク

検索エンジンがリンクの文脈を正確に理解 → PageRank評価を適切に調整

まとめ

2019年から現在にかけて、HTMLは要素間の関係性を宣言するだけで複雑なUI動作を実現できる言語へ進化

UI・インタラクション：Popover API、Dialog要素、details要素のname属性、inert属性

セマンティクス・アクセシビリティ：search要素、rel属性のSEO対応値

パフォーマンス最適化：loading属性、fetchpriority属性、blocking属性

モバイルUX：inputmode属性、enterkeyhint属性

Popover APIやdetails要素のname属性などでJavaScript依存から脱却、HTMLだけで多くのUIパターンを実現

注意

ブラウザの対応状況によってはまだ使用できない場合あり

詳細は以下を参照

- https://developer.mozilla.org/ja/docs/Web/API/Popover_API
- https://developer.mozilla.org/ja/docs/Web/HTML/Element/dialog
- https://developer.mozilla.org/ja/docs/Web/HTML/Element/details
- https://developer.mozilla.org/ja/docs/Web/HTML/Global_attributes/inert
- https://developer.mozilla.org/ja/docs/Web/HTML/Element/search
- https://developer.mozilla.org/ja/docs/Web/HTML/Element/img#loading
- https://developer.mozilla.org/ja/docs/Web/HTML/Element/img#fetchpriority
- https://developer.mozilla.org/ja/docs/Web/HTML/Element/script#blocking
- https://developer.mozilla.org/ja/docs/Web/HTML/Global_attributes/inputmode
- https://developer.mozilla.org/ja/docs/Web/HTML/Global_attributes/enterkeyhint

ご清聴ありがとうございました

本日のスライドは下記のリポジトリで公開しています。
内容の修正・改善など、お気軽にPull Requestをお送りください。

https://github.com/riya-amemiya/amemiya_riya_slide_data/tree/main/frontend_conf_kansai_2025

- XやGitHubなど: https://riya-amemiya-links.tokidux.com/