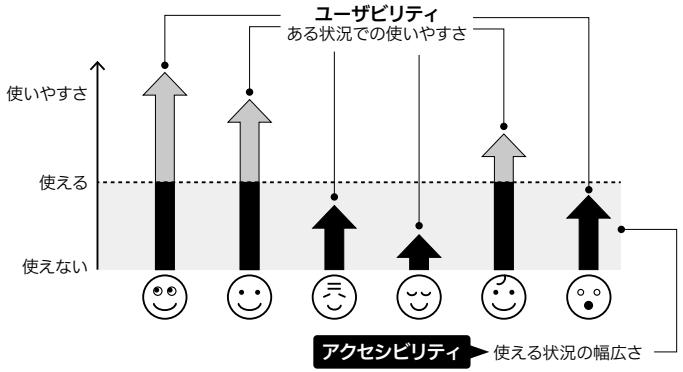
図1-1-1　ユーザビリティとアクセシビリティの関係性

『見えにくい、読みにくい「困った！」を解決するデザイン』P.23より転載（一部改変）



アクセシビリティは利用しやすさ？

アクセシビリティは「利用しやすさ」と翻訳されることもあり、辞書を引くとそのように記載されている場合もあります。しかし、この訳は誤解を生みがちです。「利用しやすさ」という語感からは「利用できることを暗黙の前提として、そのうえで使いやすいかどうかである」というイメージを持つ人が一定数存在すると考えます。そして、この語感が「改善したほうがよいけど、使えないわけではないから優先度は高くない」という解釈につながっているように思えるのです。

しかし、利用しやすさが「ゼロ」である、つまりまったく使えないことも多くあります。本書の題材であるWebアプリケーションでも、そもそもUI（*User Interface*）上にあるボタンに気付くことすらできない、キーボードフォーカスが見えずにまったく操作できない、マウスオーバーすると要素が消滅したように見えて操作できない……といった、まったく使えなくなる問題が当たり前のように潜んでいます。いずれも致命的なものであり、バグと称しても差し支えないでしょう。

アクセシビリティを高める活動とは、こうしたことが起きないように、「まず利用自体は可能である」という状況を増やしていくこと、そのための選択肢を用意していくことです。

1.2 Webアクセシビリティとは

Webにおけるアクセシビリティを考えるうえでは、まずWeb自体の特徴を理解する必要があります。誤解を恐れずにいえば、Webはアクセシビリティのために存在するメディアなのです。そう言い切れるほどに、ほかのメディアにはない「アクセスを可能にするしくみ」がWebには備わっています。そのしくみによって、あらゆる状況で利用できる可能性が開かれています。

Webにあるだけで圧倒的にアクセシブル

そもそも、Webにコンテンツやサービスがあることは、それだけでかなりアクセシブルな状態です。むしろ、Webを通じてコンテンツやサービスを提供することは、このアクセシビリティという恩恵を得たいと考えた結果といえます。

図1-2-1は、Webがそもそも持っているアクセシビリティの側面を層に分けたものです。この図でいえば、Webにコンテンツを載せて公開するだけで、堅牢・発見・携帯までがカバーできます。Webでは、あらゆるデバイスからインターネットにアクセスでき、URLによってコンテンツを一意

図1-2-1　アクセシビリティのレイヤー

図1-3-12　滞留コントロール

クリックが難しい場合、マウスポインタを一定時間、同じ場所にとどめておくとアクションが実行されるという操作方法がある。図はmacOSの滞留コントロールの操作状態

図1-3-14　音声コントロール

macOSの音声コントロールの操作状態（グリッド）

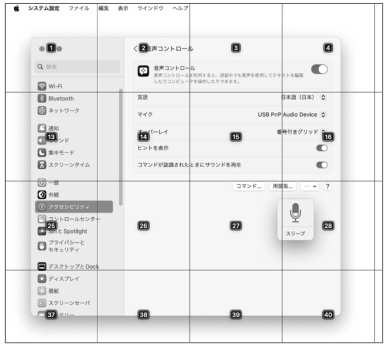


図1-3-16　柔らかいスイッチ

写真はピロースイッチ
写真提供：テクノツール株式会社

図1-3-143　音声コントロール

画面内のラベルや番号を発話することで選択したり、グリッドで区切られた領域の番号を発話してマウス相当の操作を実施したりできる。テキストの入力や修正も音声で行える。図はmacOSの音声コントロールの操作状態（項目番号）

図1-3-15　Bluetooth接続のスイッチ

写真はブルー2
写真提供：パシフィックサプライ株式会社

図1-3-17　指で挟んで押せるスイッチ

写真はフィンガースイッチ
写真提供：テクノツール株式会社

図1-3-18　センサー式のスイッチ

操作の選択と実行をすべてスイッチのオン／オフだけで行える。大きいスイッチ、弱い力でも押せるスイッチ、息で操作する呼気スイッチ、筋肉の動きを検知する筋電位スイッチなど、さまざまなものがある。写真は、わずかな筋肉の動きで作動するピエゾセンサー（圧電素子）と、指先の僅かな動きで作動するエアバッグセンサー（空圧）の2種類が付くピエゾニューマティックセンサスイッチ
写真提供：パシフィックサプライ株式会社

図1-3-19　macOSのスイッチコントロール（ホーム）

スイッチコントロールのメニューから、マウスポインタ操作モードを選択する

図1-3-20　macOSのスイッチコントロール（ポインタ）

ポインタをどう動かすかを選ぶ。ここでは移動してクリックを選択する

図1-3-21　macOSのスイッチコントロール（グライドカーソル）

X座標・Y座標をクレーンゲームのように指定すると、マウスポインタが移動してクリックが実行される

良い例：HTML標準のチェックボックス

```
<label>
  <input type="checkbox" checked="checked" />
  同意する
</label>
```

input要素はtype属性ごとに異なる役割を持ちます。type="checkbox"は、チェックボックスの役割を持っています。またlabel要素によって「同意する」のコンテンツと関連付けられ、「同意する」という名前を持ちます。これにより同意するかしないかの真偽値を入力するチェックボックスであることがわかります。

またチェックボックスがチェックされていると、真偽値を持つchecked属性にcheckedの値が入り、チェックされている状態であることを示します。

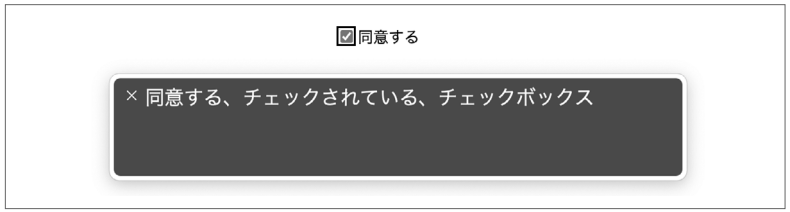
ためしにスクリーンリーダーでチェックボックスを読み上げてみると、ラベルの「同意する」、状態として「チェックされている」、そして役割の「チェックボックス」が読み上げられます(**図2-1-4**)。

現在の値がわからなければ、操作ができたとしてもチェックできたかどうかがわからず、操作の目的は果たせません。何かを入力するUIには「名前」と「役割」に加えて「状態」が必要です。

HTMLのセマンティクスと、それを補完するWAI-ARIA

さて先の例で、セマンティクスを実装することがアクセシビリティにとって大事だと述べました。たとえば要素に「ボタン」というセマンティクスを持たせることによって、ユーザーはその要素が実行可能であることがわかりますし、「チェックされている」という状態を持たせることによって、入力されているかどうかがわかります。

図2-1-4　「同意する」ラベルがあるチェックボックスのスクリーンリーダーによる読み上げ



HTMLはセマンティクスを持っている

読み上げられるボタンの実装例では、button要素を使用しました。button要素はボタンという役割を持っています。これはHTMLのネイティブセマンティクスと呼ばれるものです。

基本的にはHTMLの要素を適切に使用することでセマンティクスを実装できます。ただしHTMLの要素や属性には限りがあります。ドキュメントを共有する目的から始まったHTMLの語彙は、アプリケーション独特のセマンティクスが不足しています。たとえばタブなどのインタラクションを含む役割や、要素が展開されているかどうかという状態です。

HTMLのセマンティクスを補完するWAI-ARIA

これらのセマンティクスを補完するのがWAI-ARIAという仕様です。WAI-ARIAには主に役割を補完するWAI-ARIAロールと、状態やプロパティを補完するWAI-ARIAステートおよびプロパティがあります。

もともとWAI-ARIAは、HTMLにないセマンティクスを補完するために誕生しました。たとえばタブUIは、HTMLだけではタブUIであること、タブが選択されていること、タブのコンテンツが隠されているかどうかなどのセマンティクスを表現できませんでしたが、WAI-ARIAを使うと表現することが可能です(**図2-1-5**)。

図2-1-5　WAI-ARIAを用いて実装されたタブUIのタブ

Google Chromeの開発者ツールによる表示(以下同)

よくある事例で課題を知る

[事例1]エラーの発生箇所とエラーメッセージの関係がわかりづらい

エラーの発生箇所とエラーメッセージの関係がわかりづらい場合があります。

たとえば、エラーの発生箇所とエラーメッセージが離れて配置されていることがあります。特に画面を拡大しているユーザーは、エラーの発生箇所とエラーメッセージを同時に視認できないことがあるため、エラーがどこで発生しているのかわかりにくくなります(**図3-3-1**)。

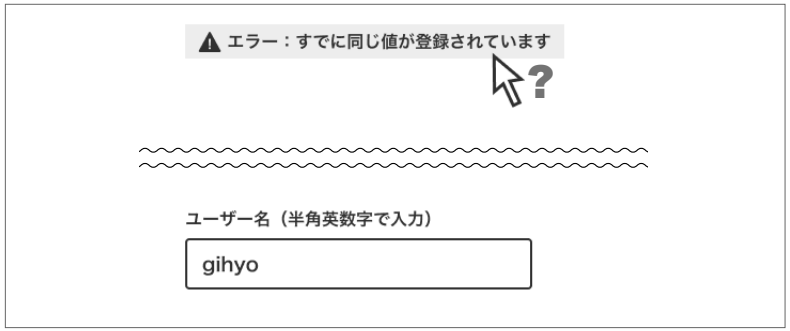
ほかにも、エラーの発生箇所とエラーメッセージの関係が適切にマークアップされていない場合もあります。たとえば、以下ではエラーが適切にマークアップされておらず、エラーメッセージがフォームコントロールに関連付けられていません。

悪い例：エラーが適切にマークアップされていないフォームコントロール

```
<label for="username">
  <span>ユーザー名（半角英数字で入力）</span>
</label>
<span>エラー：すでに同じユーザー名が登録されています。</span>
<input type="text" id="username">
```

エラーが関連付けられるようにマークアップされていないと、スクリーンリーダーはエラーの発生箇所にたどり着いてもエラーメッセージを読み上げません。スクリーンリーダーを利用しているユーザーは、エラーが発生していることがわからない恐れがあります。

図3-3-1　エラーの発生箇所とエラーメッセージが離れて配置されている



[事例2]多様なユーザーへエラーを通知する方法を検討していない

多様なユーザーへエラーを通知する方法が検討されていないと、ユーザーによってはエラーの発生やエラーの発生箇所を理解できない恐れがあります。フォームを送信したときに、エラーが発生した箇所の近くに単にエラーを表示しただけでは、エラーが適切に通知されているとはいえません(**図3-3-2**)。

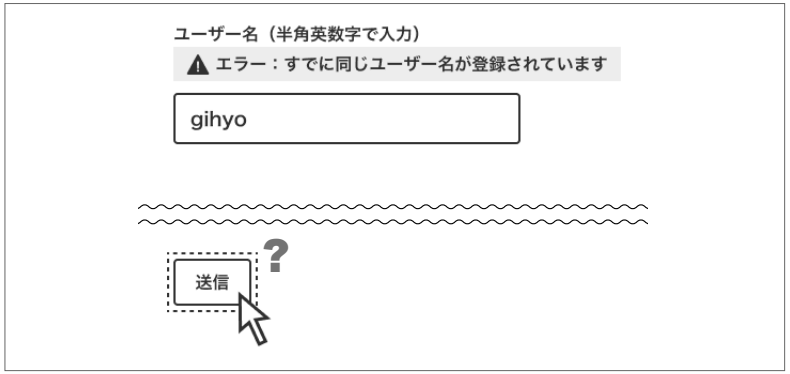
たとえば、スクリーンリーダーを利用するユーザーはエラーが発生したことに気付けない可能性があります。スクリーンリーダーは基本的にフォーカスしている箇所を読み上げます。送信ボタンを押してクライアントエラーが発生したときには送信ボタンにフォーカスしています。したがって送信ボタンとは別の箇所にエラーメッセージが表示されても、そのままではエラーメッセージが読み上げられません。

また、画面を拡大しているユーザーもエラーが発生したことに気付けない可能性があります。画面を拡大していて画面の一部しか視認できていないと、エラーメッセージが画面におさまっていなければエラーメッセージを見落としてしまうからです。

[事例3]エラーの修正方法がわかりにくい

エラーの修正方法がわかりにくいと、ユーザーはエラーを通知されてもエラーを修正できません。わかりにくいエラーメッセージには典型的なパターンがあります。

図3-3-2　多様なユーザーへエラーが通知されない



グラフの例も、色以外に形状を利用して改善します。折れ線グラフであれば、線を破線にする、マーカーの形状を変えるなどにより、色に依存せずに情報を提供できます(**図4-1-7**)。

そのほか、凡例とデータの対応を直接線で結ぶ方法があります。また形状で示す以外にも、「赤」と「緑」の組み合わせを避けるなど、さまざまな色覚でも見分けやすい配色を検討することも大事です(**図4-1-8**)。

色のみに依存することに問題があるだけで、ユーザーが情報の分類を理解するのを助けたり、イメージを喚起したりと、色が強力なツールであることは間違いありません。情報提供を色のみに依存しないようにしたうえで、色をうまく活用することを目指しましょう。

[事例2の改善❶]文字のコントラスト比を改善する

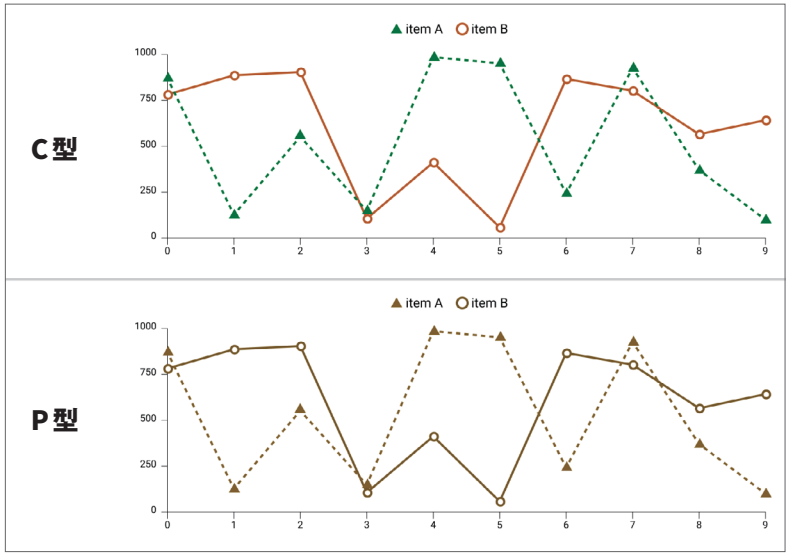
WCAG 2.1の達成基準 1.4.3「色のコントラスト(最低限)」[注3]では、文字色と背景色のコントラスト比を4.5：1以上にすることを求めています(**図4-1-9**)。

より発展的な達成基準 1.4.6「色のコントラスト(高度)」[注4]では、文字色と

注3　https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html
日本語訳：https://waic.jp/docs/WCAG21/Understanding/contrast-minimum.html

図4-1-7　折れ線グラフの改善例

データと判例の紐付けが色だけでなく、線のスタイルやマーカーの形状でもされている



注4　https://www.w3.org/WAI/WCAG21/Understanding/contrast-enhanced.html
日本語訳：https://waic.jp/docs/WCAG21/Understanding/contrast-enhanced.html

図4-1-8　折れ線グラフの改善例

凡例とデータを線で結び、配色も見分けやすくしている

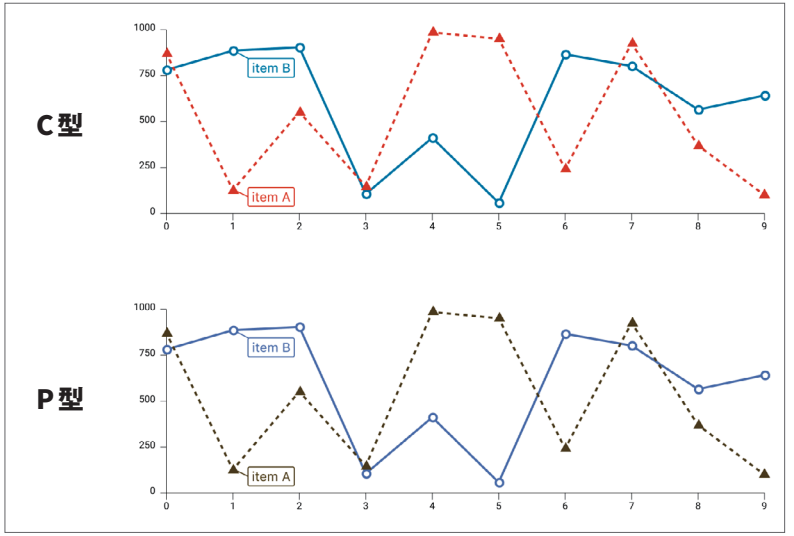
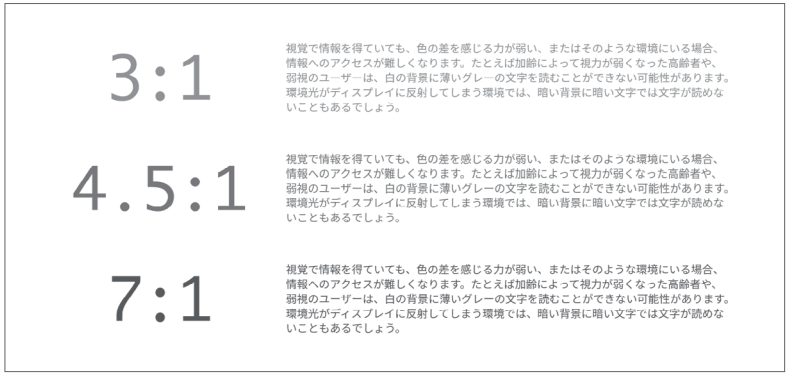


図4-1-9　3:1、4.5:1、7:1のコントラスト比のサンプル

白の背景に黒もしくはグレーの文字で示されている



5.1 モーダルダイアログ

本節ではモーダルダイアログを取り上げます。モーダルダイアログまたは単にモーダルと呼ばれる機能は、ユーザーへの説明や確認、フォームの表示など、モダンなWebアプリケーションには必ずといってよいほど存在します。

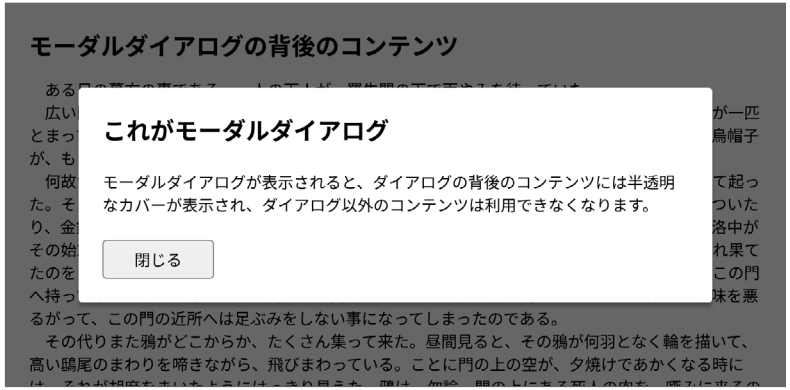
モーダルダイアログは、以下のような特徴を持ちます。

- **ページの描画よりあとに、ユーザーの操作やアプリケーションのロジックによって表示される**
- **ほかのコンテンツの上に被さるように表示され、表示されている間はモーダルダイアログ内の要素のみ操作できる**
- **モーダルダイアログ内での操作によって非表示になる(閉じられる)ことがある**

Webページにおけるモーダルダイアログは多くの場合、ネイティブアプリケーションのダイアログを模した小さなウィンドウ状の矩形(くけい)領域として表示されます。半透明の要素で全画面を覆い、その上に重ねて表示することもあります(**図5-1-1**)。

図5-1-1　モーダルダイアログ

背後のコンテンツは半透明の要素で隠されている



「モーダル」とは、一時的にシステムが通常と違うモードになって、限定的な操作だけを受け付ける状態を指します。通常のWebページは「モードレス」で、ページ内のいろいろな部分を行き来できるのに対し、「モーダル」なダイアログはそのダイアログという「モード」にユーザーを閉じ込めてしまいます。

モーダルダイアログが表示されると、ユーザーはその外のコンテンツを操作できなくなります。表示されている間はユーザーのできることが限定されてしまいますが、特定の順序に沿って情報を確認したり操作したりする必要があるときには有効な手段です。

モーダルダイアログの例と問題点

モーダルダイアログはページ内のほかの要素に被せて表示するため、しばしばbody要素の最後に要素を差し込む形で実装されます。body直下に配置することで、親要素のスタイルの影響を受けないようにしつつ、最後に挿入することで同じz-indexを持つ要素が存在しても最前面に表示されます。

以下のコードは、モーダルダイアログをReactによって実装した例です。

表示すると「ダイアログを開く」ボタンと「技術評論社のWebサイト」のリンクが置かれ、モーダルダイアログには「ダイアログを閉じる」ボタンが配置されています(**図5-1-2**)。

悪い例：アクセシビリティを考慮せず作られたモーダルダイアログのコンポーネント（TypeScript）

```
import { useEffect, useState, useRef } from 'react';
import { createPortal } from 'react-dom';
```

図5-1-2　モーダルダイアログの表示例

5.4 シンプルなツールチップ

本節ではツールチップについて紹介します。本節では、特定の場所にマウスオーバーすることによって、マウスポインタの付近に表示されるもの全般をツールチップと呼びます。

「対象となる要素へのマウスオーバーによって何かを表示する」という単純な機能ひとつでも、アクセシビリティに関して多くの問題を生みます。たとえば、ツールチップの内容を読めるようにするには、画面を拡大しての使用や、キーボードでの操作、スマートフォンやタブレットでの使用にそれぞれ別々の工夫がいります。

すべての問題をひとつの方法で解決できる定石はありません。作ろうとしているものについて起き得る問題をきちんと認識し、ひとつひとつどう対処していくのかを考えなければなりません。

本節では、内容がテキストのみの「シンプルな」ツールチップについて紹介します。内部にリンクなどのインタラクティブな要素や複雑な構造を含んでいたり、自然な表示になるようにDOM上の離れた箇所に挿入されたりしている、「リッチな」ツールチップについては次節で扱います。ツールチップの要件によっては次節の内容まで考慮に入れて検討する必要があります。あわせて参照してください。

シンプルなツールチップの例と問題点

最も簡単なツールチップの実装は、title属性を使用するものでしょう（表示例は**図5-4-1**）。

例：リンクにtitle属性でツールチップを付加

```
<a href="https://gihyo.jp/" title="この本を出版している会社のWebサイト">技術評論社のWebサイト</a>
```

title属性によるツールチップは簡単に実装できます[注11]。しかし、ツールチップが表示されるタイミングを制御したり、ツールチップの見た目を変えたり、多機能化したりすることはできません。

そこで、多くのWebサイトやサービスでは、ツールチップのような見た目と挙動をするもの、すなわち特定の場所にマウスオーバーすることでその付近に表示されるものを独自に実装しています。たいていの場合、ツールチップの対象となる要素の位置の近くに、CSSでposition: absoluteを指定して配置しています。

以下はReactでツールチップを独自実装した例です（表示例は**図5-4-2**）。この実装には、次のような問題があります。

- **画面拡大時にツールチップを読めない**
- **キーボード操作ではツールチップを読めない**
- **スマートフォンやタブレットではツールチップに気が付かない**
- **スクリーンリーダーではツールチップに気が付かない**

本節では順番に改善方法を紹介します。

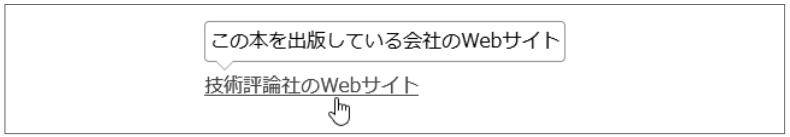
悪い例：アクセシビリティを考慮せず作られたツールチップのコンポーネント（TypeScript）

```
import { useState } from 'react';
const TooltipSample = () => {
  // ツールチップの表示状態を保持するstate
  const [isTooltipVisible, setTooltipVisible] = useState(false);

  return <span className="withTooltip">
    <a href="https://gihyo.jp/"
      // mouseenterでツールチップを表示し、mouseleaveで非表示にする
      onMouseEnter={() => setTooltipVisible(true)}
```

図5-4-1 title属性によるツールチップ

図5-4-2 独自実装のツールチップの表示例



注11 HTMLの仕様には、title属性への依存は推奨しないと記載しています。この記載の主旨は本節の内容とも重なるので、詳細はのちほど解説します。

6.1 デザインシステムとは

デザインシステムは、広い意味でのデザインの一貫性を保つための、ドキュメントや部品、ガイドライン、制作プロセスなどの総称です。

組織の中で一貫性のあるデザインを実現するには、そのデザインがどのようなルールに基づいて、どのように作られるべきかを明文化することが重要です。本章では、これらを体系的に整備したものをデザインシステムと呼びます。

デザインシステムの構成

デザインシステムには多くの場合、以下のものが含まれています。

- **デザイン原則**
- **スタイルガイド**
- **パターンライブラリ**

デザイン原則には、製品やそれを作る組織が重視する価値観、判断基準が定義されます(**図6-1-1**)。製品のデザインとして考慮し、目指すもののほか、組織によってはブランドとして表現したいもの(ブランドアイデンティティ)なども含まれてきます。そして、その原則をもとにして、スタイルガイドやパターンライブラリなどが策定されます。

スタイルガイドには、製品が使用するアイコンやタイポグラフィ、カラーパレット、文言のガイドラインなどが含まれます(**図6-1-2**)。デザイン原則やブランドアイデンティティに沿って、どのようなものを組み合わせてデザインを作っていけばよいのかを示します。カラーパレットなどは、デザイントークンと呼ばれる形でCSSの変数として定義されることもしばしばあります。

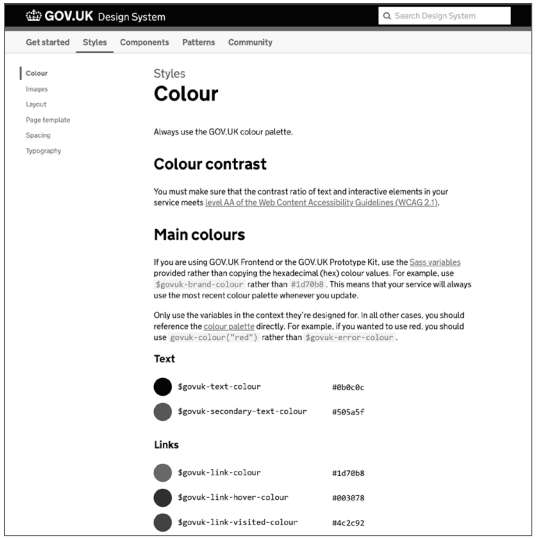
パターンライブラリは、デザインを構成するデザイン言語をパターン化し、ライブラリとして再利用可能にしたものです(**図6-1-3**)。製品が提供する機能であったり、ユーザーへの見せ方であったりと、パターンの粒度はさまざまです。より細かい粒度でUIコンポーネントのライブラリとなっ

図6-1-1 デザイン原則の例

SmartHR Design Systemのデザイン原則
https://smarthr.design/foundation/

図6-1-2 スタイルガイドの例

GOV.UK Design SystemのColour
https://design-system.service.gov.uk/styles/colour/



いった話を続けられることが、アクセシビリティをおもしろいと感じる原動力となり、長い目で見るとデザインやエンジニアリングに役立ちます。

こうしたサロンはランチやおやつタイムと合わせて実施されています。freeeでは週1回、ヤクルトが届く午後の時間に合わせて開催していました[注7]。

7.4 自身の考えを社内で発信する

情報を取得し、アクセシビリティに対する感覚がつかめたら、次はアウトプットです。現時点ではまだ興味を持った人たちによる、一種の「部活」

注7　本書執筆時点（2022年12月）ではCOVID-19の影響により、オンライン開催に移行しています。

図7-3-3　サロンの開催を伝えるホワイトボード

堅苦しくならないように手書きでゆるい感じを演出し「誰でも参加OK　お菓子もあるよ」とも書き添えている

の状態です。会社公認の活動にしていくには、アウトプットに取り組む必要があります。

社内向けにアクセシビリティの記事を書く

社内向けWikiなど[注8]があれば記事を書いてみましょう。

内容は何でもかまいません。輪読会のメモや、自分なりのスクリーンリーダーの使い方など、前節での活動を実施するたびに、その記録や気付いたことのメモなどを投稿するのが手軽です。そのほか、既存のアクセシビリティ関係の資料の要約や、イベント参加レポートなども有用です。

書いておけば誰かの役に立つかもしれないと一瞬でも思えたら、少しずつでも投稿します。アウトプットを重ねることで、自分が取り組んでいる理由や、フォーカスしたいと思っている領域などの整理が進みます。

きちんとしたものを書こうとすると、どんどん時間が過ぎます。あなたがあえてハードルの低い記事を投げ込んでいくことで、ほかの人もアクセシビリティにまつわる話を投稿してもよさそうな空気を作れます。短くても拙くても投稿を少しずつ積み上げましょう。

社内の発表会でアクセシビリティの話をする

この時点では、アクセシビリティに関する活動はまだ限られた人にしか伝わっていません。あなたの活動にリアクションしてくれる人たちが何人か出てきていても、それはおそらく「もともとアクセシビリティにある程度関心があった人」なのです。

人間の認知のリソースは限られており、自分とあまり関係ないと思ったものはスルーします。そこに割り込むには、相手がアクセスしてくれるのを待つのではなく、こちらからプレゼンテーションしていく必要があるのです。社内でのライトニングトークや取り組み発表の機会があれば、積極的に手を挙げましょう（**図7-4-1**）。

プレゼンテーションの内容は「取り組みの現状報告」でよいでしょう。今

注8　Scrapbox、Confluence、Qiita Team、esa、kibela、Notionなどです。

8.3 アンチパターンと対策❶ ——1画面に多くの状態を持つ

操作対象が散らばり、状態把握が難しくなる

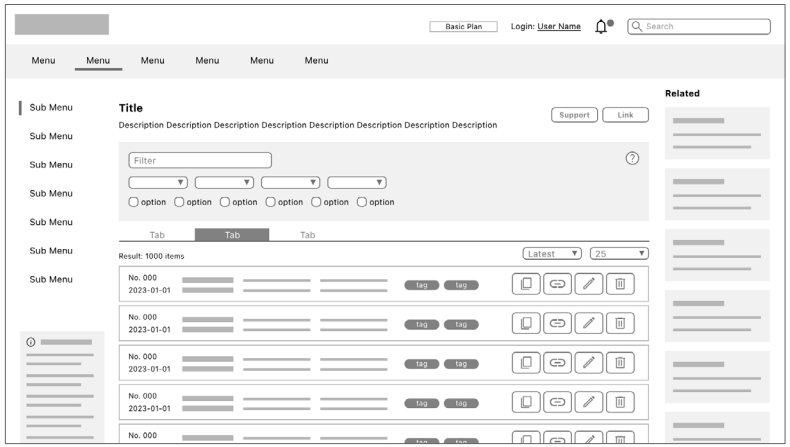
多くのWebアプリケーションのレイアウトは以下の構成になっており、1画面内に操作対象が散らばり、多数の状態の組み合わせが存在します(**図8-3-1**)。

- **ヘッダ、上や左のナビゲーションバー、メインエリアの上部・中央・下部、サブエリア、モーダルダイアログといったさまざまなエリアがある**
- **エリアごとに、ボタンやリンク、フォームコントロールといったインタラクティブ要素がある**
- **インタラクティブ要素の操作結果として、ナビゲーションの選択状態、表示内容の切り替え状態、編集モードなどが存在する**

マウス操作が困難な場合、多くの操作対象が画面内に散らばるほどに負担も増えます。どこかを選んで、何かを見て、スクロールして、何かを切り替えて、また戻って……という手順自体に大きなコストがかかります。

図8-3-1　アプリケーション画面の例

さまざまなナビゲーションとリンクとボタンが複雑に組み合わさっている



キーボードのみで操作する場合も同様です。マウス操作と違い、キーボード操作では画面内の領域をまたぐような非連続な移動ができず、Tabキーによる連続的なフォーカス移動でしか操作対象の切り替えが行えないからです。

画面を大きく拡大しているときやスクリーンリーダー利用時も、この「散らばった操作対象の行き来」には同様の手間がかかります。現在マウスポインタやスクリーンリーダーカーソルがある場所しか認識できない[注6]ので、行き来をするためには毎回ポインタやカーソルを動かして操作対象を探すことになります。さらに、推定と記憶に頼らざるを得ないという問題も生じます。操作結果が視界の外に反映されて気付けなかったり、視界から外れた現在位置表示の状態をあとで思い出す必要が生じたりするからです。

多くの操作対象が画面内に散らばり、状態把握が難しいWebアプリケーションが増えている理由としては、以下の点が考えられます。

- **切り替えウィジェットの濫用**
- **複数ペイン構成の濫用**
- **モーダルダイアログの濫用**

ここからは、この3点について詳しく考察していきます。

切り替えウィジェットの濫用

アンチパターン

切り替えウィジェットは、画面上に存在する要素を一時的に隠しておき、要求があったときに表示するUIパターンです。ユーザーとのインタラクションによって内容が変化することを表現するためによく用いられます。一般的なUIパターンですが、1画面に要素を押し込みすぎて複雑になってしまった「切り替えウィジェットの使いすぎ」といえる状況も見かけるようになりました。

例としては、アコーディオン型のフォームが挙げられます(**図8-3-2**)。ECサイトでの購入フローなどにおいて、全体がアコーディオン形式になっ

注6　aria-live属性を使うなどして、変化を明示的に伝えている場合は除きます。