

JSON-HTML変換モジュール

概要

`converter.js` は記事のJSONデータをHTMLに変換するモジュールです。記事の表示時に各コンテンツブロック（見出し、段落、リスト、画像など）をスタイル付きのHTMLに変換し、一貫したデザインで表示します。

基本構造

このモジュールは以下の主要コンポーネントで構成されています：

1. **スタイル定義** - 各要素のCSSスタイル
2. **インラインフォーマット定義** - テキスト内のマークダウン風書式処理
3. **変換関数** - 記事とブロックをHTMLに変換
4. **ユーティリティ関数** - HTMLエスケープや日付フォーマットなど

API関数

記事変換 (`articleToHtml`)

記事オブジェクトをHTMLに変換します。

```
// 使用例
const article = {
  id: 'article_1',
  title: '記事タイトル',
  category: 'カテゴリ',
  tags: ['タグ1', 'タグ2'],
  updatedAt: '2023-04-01T12:34:56Z',
  content: [
    { type: 'heading', text: '見出し', level: 2 },
    { type: 'paragraph', text: '本文テキスト' },
    // ...他のブロック
  ]
};

const html = Converter.articleToHtml(article);
document.getElementById('article-container').innerHTML = html;
```

パラメータ

- article (Object) - 記事オブジェクト

戻り値

- string - HTML文字列

内部実装の詳細

スタイル定義

モジュールは共通スタイルをJavaScriptオブジェクトとして定義しています。これにより、HTML出力時に一貫したデザインを実現します。

```
const STYLES = {
  headingH2: 'font-size: 22px; color: #2d3748; font-weight: 700; margin: 0; padding-bot
  paragraph: 'margin: 0; font-size: 16px; line-height: 1.8; color: #4a5568; text-align:
  // ...他のスタイル定義
};
```

インラインフォーマット

テキスト内のマークダウン風構文を対応するHTML要素に変換します。

```
const INLINE_FORMATS = [
  { pattern: /__([^_]+?)__/g, replacement: '<u style="text-decoration: underline;">$1</u>' },
  { pattern: /\*\*([^*]+?)\*/g, replacement: '<strong style="font-weight: 700;">$1</strong>' },
  { pattern: `/`(([^`]+?)`)/g, replacement: '<code style="font-family: monospace; background-color: #f0f0f0;">$1</code>' },
  { pattern: /_([^_]+?)_/g, replacement: '<em style="font-style: italic; color: #4a5568;">$1</em>' },
  { pattern: /==([^=]+?)==/g, replacement: '<span style="color: #3182ce;">$1</span>' },
  { pattern: /!!([^!]+?)!!/g, replacement: '<span style="color: #e53e3e;">$1</span>' },
  { pattern: /@@([^@]+?)@@/g, replacement: '<span style="color: #38a169;">$1</span>' },
  { pattern: /\+\+(?:[^+]+?)\+\+/g, replacement: '<mark style="background-color: #fefcbf; border: 2px solid #e5c494;">$1</mark>' }
];
```

サポートされる全インラインフォーマット：

- **太字** → 太字
- _斜体_ → 斜体
- `コード` → コード
- __下線__ → 下線
- ==青色== → 青色テキスト
- !!赤色!! → 赤色テキスト
- @@緑色@@ → 緑色テキスト
- ++ハイライト++ → ハイライト

ブロック変換関数

記事内の各ブロックタイプに対応する変換関数があります。

見出し変換 (renderHeadingBlock)

見出しブロックをHTMLに変換します。level属性によってH2～H6タグが生成されます。

```
function renderHeadingBlock(block) {
  const level = block.level || 2;
  const text = processInlineFormatting(block.text || '');

  if (level === 2) {
    return `<div style="margin: 30px 0 16px 0;"><h2 style="${STYLES.headingH2}">${text}`;
  } else if (level === 3) {
    return `<div style="margin: 25px 0 14px 0;"><h3 style="${STYLES.headingH3}">${text}`;
  } else {
    // ...レベルに合わせたスタイル
  }
}
```

段落変換 (renderParagraphBlock)

段落ブロックをHTMLに変換します。

```
function renderParagraphBlock(block) {
  return `<div style="margin-bottom: 18px;"><p style="${STYLES.paragraph}">${processInl
}
```

リスト変換 (renderListBlock)

リストブロックをHTMLに変換します。ordered（番号付き）とunordered（箇条書き）の2つのスタイルをサポートしています。さらに、ネストされたリスト（サブ項目）もサポートしています。

```

function renderList(items, style, nestLevel = 0) {
  // 項目が空の場合は空文字列を返す
  if (items.length === 0) return '';

  const isOrdered = style === 'ordered';
  const listItems = items.map((item, index) => {
    // ネストされたリストがあるか確認
    const hasNestedList = item.nestedItems && item.nestedItems.length > 0;
    const itemText = processInlineFormatting(item.text || item);

    // ネストレベルに応じた色を選択（薄めの色）
    const markerColor = nestLevel > 0 ? '#94a3b8' : '#4a5568';

    // ...マーカーと項目を生成

    // 再帰的にネストされたリストを処理
    const nestedListHtml = hasNestedList
      ? renderList(item.nestedItems, item.nestedStyle || style, nestLevel + 1)
      : '';

    // ...リスト項目のHTMLを生成
  }).join('');

  // ...リスト全体のHTMLを生成
}

```

ネストされたリストの処理:

- リスト項目内にネストされたリスト（`nestedItems`）がある場合、再帰的に処理
- ネスト階層に応じてマーカーの色を調整（親項目は濃い色、子項目は薄い色）
- ネストレベルに応じたマージンとパディングを適用し、視覚的な階層を表現

コードブロック処理

コードブロックはマークダウンの“”で囲まれたテキストを変換します。視覚的に区別しやすいよう、暗い背景色と明るいテキスト色が適用されます。

```

// コードブロック変換
html = html.replace(/\````([^\"]+)````/g, '<pre style="background-color: #1a202c; padding:

```

インラインコードは境界線を追加して視覚的な区別を強化しています:

```
{ pattern: `/`([^\`]+?)`/g, replacement: '<code style="font-family: monospace; background
```

画像変換 (renderImageBlock)

画像ブロックをHTMLに変換します。マークダウン形式の ![代替テキスト](画像URL) 構文もサポートしています。

```
function renderImageBlock(block) {
  const src = escapeHtml(block.src || '');
  if (!src) return '';

  const alt = escapeHtml(block.alt || '');
  const caption = block.caption ? processInlineFormatting(block.caption) : '';

  // サイズ指定の処理
  let sizeStyle = '';
  // ...width/height処理

  return `
    <div style="${STYLES.imageContainer}">
      <figure style="margin: 0; padding: 0;">
        
        ${caption ? `<figcaption style="${STYLES.imageCaption}">${caption}</figcaption>` : ''}
      </figure>
    </div>
  `;
}
```

マークダウン形式での画像挿入例：

```
![画像の説明](画像のURL)
```

この構文は、マークダウンテキスト処理時に画像コンテナに変換され、適切なスタイルが適用されます。アライメント（左・中央・右）や幅・高さのカスタマイズも可能です。画像幅と高さはピクセル値（例: 300）またはCSS値（例: '50%'）として指定できます。

ユーティリティ関数

インラインフォーマット処理 (processInlineFormatting)

テキスト内のマークダウン風構文をHTMLに変換します。

```
function processInlineFormatting(text) {
  if (!text) return '';

  // 安全のためにHTMLをエスケープ
  let processed = escapeHtml(text);

  // 各パターンを処理
  INLINE_FORMATS.forEach(format => {
    processed = processed.replace(format.pattern, format.replacement);
  });

  return processed;
}
```

現在の実装では、インライン記法の組み合わせ（例：「太字の中のイタリック文字」）には完全には対応していません。これは将来の改善点として以下のTODOが記録されています：

```
// TODO: インライン記法の組み合わせ対応
// - 「**太字の中の_イタリック_文字**」のような組み合わせに対応する
// - 内部から外部への処理順序の改善または再帰的処理の実装が必要
// - パターンの順序にも注意して実装する
```

HTMLエスケープ (escapeHtml)

HTML特殊文字をエスケープします。

```
function escapeHtml(text) {
  if (!text) return '';
  const escapeMap = {
    '&': '&amp;',
    '<': '&lt;',
    '>': '&gt;',
    '\"': '&quot;',
    '\''': '&#039;'
  };
  return String(text).replace(/[\&<>\"]'/g, match => escapeMap[match]);
}
```

日付フォーマット (formatDate)

ISO形式の日付文字列を読みやすい形式に変換します。

```
function formatDate(dateString) {
  try {
    const date = new Date(dateString);
    return date.toLocaleDateString('ja-JP', {
      year: 'numeric',
      month: 'long',
      day: 'numeric'
    });
  } catch (e) {
    return dateString || '';
  }
}
```

出力例

このモジュールによって生成されるHTMLの例：

```

<article style="border: 1px solid #e8e8e8; border-radius: 8px; padding: 25px; box-shadow: 0 0 10px #e8e8e8; margin-bottom: 20px; background-color: #f8fafc; position: relative; z-index: 1;">
  <header style="margin-bottom: 35px; border-bottom: none; padding-bottom: 0; position: relative; z-index: 1;">
    <h1 style="margin: 0 0 15px 0; font-size: 30px; color: #2d3748; font-weight: 600; line-height: 1.2; position: relative; z-index: 1;">
      <div style="font-size: 14px; color: #666; margin-bottom: 20px; display: flex; align-items: center; justify-content: space-between; position: relative; z-index: 1;">
        <div style="display: flex; align-items: center; flex-wrap: wrap; position: relative; z-index: 1;">
          <span style="display: inline-block; background-color: #edf2f7; color: #4a5568; width: 15px; height: 15px; border-radius: 50%; margin-right: 10px;">
          <span style="display: inline-block; background-color: #e6f6ff; color: #2b6cb0; width: 15px; height: 15px; border-radius: 50%; margin-right: 10px;">
        </div>
        <span style="color: #718096; position: relative; z-index: 1;">最終更新: 2023年4月1日
      </div>
    </h1>
  </header>

  <div class="article-content">
    <div style="margin: 30px 0 16px 0;"><h2 style="font-size: 22px; color: #2d3748; font-weight: 600; margin-bottom: 10px; position: relative; z-index: 1;"><!-- 他のコンテンツ --><!-- 他のコンテンツ -->

```

注意点

- このモジュールはJSONデータからHTMLへの一方向変換のみを提供します。逆変換（HTMLからJSON）は提供していません。
- 変換されたHTMLにはインラインスタイルが含まれており、外部CSSに依存せずに一貫した表示が可能です。
- テキスト内容は自動的にHTMLエスケープされるため、安全にHTMLとして出力できます。
- プレーンなHTMLではなく、すべての要素にスタイルが適用されています。
- 大量のデータや長い記事を扱う場合、メモリ使用量と処理時間に注意してください。
- ネストリスト（サブ項目）はサポートされていますが、視覚的区別は最小限に留められています。親項目は濃い色のマーカー、子項目は薄い色のマーカーで表示されます。
- コードブロックとインラインコードは視覚的に区別されています。コードブロックは暗い背景（#1a202c）に明るいテキスト（#e2e8f0）、インラインコードは薄い背景（#f1f5f9）に青いテキスト（#1e40af）と境界線を使用しています。
- インライン記法の組み合わせ（例：「**太字の中の_イタリック_文字**」）は現在完全にはサポートされていません。これは今後の課題として記録されています。

名前空間へのエクスポート

```
// 名前空間の初期化
window.KB = window.KB || {};
window.KB.converter = window.KB.converter || {};

// コンバータモジュールを名前空間に追加
window.KB.converter = Converter;
window.Converter = Converter; // 互換性のため
```

このモジュールは、グローバルの `window.KB` 名前空間の `converter` プロパティとして機能をエクスポートしています。また、後方互換性のために `window.Converter` としても公開されています。

アプリケーション全体で一貫した名前空間パターンを使用することで、モジュール間の連携が容易になり、グローバル名前空間の汚染を防ぎます。