

新ダッシュボードのリリースに伴うカスタムコードへの影響について

概要

新ダッシュボードのリリースに伴い、ダッシュボードの仕様変更が行われた影響で、旧ダッシュボード用のカスタムコードが正常に動作しなくなる可能性があります。

本記事では、旧ダッシュボード用のカスタムコードをご利用中のお客様向けに、カスタムコードの動作に影響する可能性がある仕様変更について記載しています。新ダッシュボードへの移行時にご参照ください。

新ダッシュボード機能の基本的な使い方は[こちら](#)の記事をご参照ください。

対象

ご利用中のカスタムコードの内、適用画面が「ダッシュボード」となっているカスタムコード（JavaScriptまたはCSS）が対象です。

レコード詳細、一覧、登録、完了画面に適用しているカスタムコードは、ダッシュボードの切り替えによる変更はございません。

注意点

- ダッシュボード用カスタムコードが新ダッシュボードのリリース前から有効で、旧ダッシュボードで使用していたパネルの設定が残っているアプリは、初期表示の時点では旧版が表示されます。その他のアプリは新版のダッシュボードが表示されます。
- 旧ダッシュボードはパネルの表示はこれまでどおり動作しますが、パネルの追加、修正、並び替え、表示・非表示の切り替え等の操作ができません。**
- 新旧の切り替えはアプリ単位で行います。切り替えると、そのアプリを閲覧・操作する全員の表示が変わります。

- 新ダッシュボードへの切り替えは「アプリの作成者」または「アプリの設定権限を持つ開発者アカウント」だけが行えます。
- 新旧の表示の切り替えは何度でも行えますが、「**新しいダッシュボードで確定する**」をクリックすると、旧版表示には戻すことができませんのでご注意ください。
- **カスタムコードの変更履歴はアプリ上に残りません**。修正前のカスタムコードはお客様自身で管理していただくようお願いいたします。
- 旧・新ダッシュボードの両方に同じカスタムコードが反映されます。

ご利用中のカスタムコードの動作に影響が出る可能性がある仕様変更

こちらのセクションでは、ご利用中のカスタムコードの動作に影響が出る可能性がある仕様変更をまとめています。

ご利用中のカスタムコードに該当する処理が無いかご確認ください。

なお、**カスタムコードの変更は旧・新版の両方に反映されます**。新版用の処理に変えた場合、**旧版では正常に動作しない可能性があります**のでご注意ください。

※参考：[5. 旧版の動作に影響しないコードの書き方](#)

1. パネルの挿入タイミング

旧版ではパネルのHTMLは最初からページに含まれていましたが、新版では画面の読み込み後にJavaScriptがサーバーから取得して挿入します。そのため、次のような書き方は新版ではパネルがパネルが挿入される前に実行されてしまいます。

- `$(function(){ ... })` や `DOMContentLoaded` の中でパネルを触る処理
- `$(document).ajaxComplete(...)` や `ajaxStop` でパネルの読み込み完了を待つ処理（パネルの取得はjQueryのajaxではないため、これらは発火しません）

新版では、パネルの描画が終わると `document` に `dashboardReady` というイベントが発生します。パネルを変更する処理はこのイベントで実行してください。

【例】

```
document.addEventListener('dashboardReady', function (e) {  
    $('#dashboard .card').each(function () { /* ... */ });  
});
```

```
});
```

【`dashboardReady` から受け取ることができるプロパティ例】

パネルの配置を操作するオブジェクト（GridStack）。

`e.detail.grid`

`grid.update (...)` などで位置やサイズなどを変更できる。

`e.detail.items`

描画したパネルの一覧（配列）。各要素に `id`, `x`, `y`, `width`, `height`, `content` を持つ

`e.detail.reload`

パネル1枚だけを読み直したときだけ `true`（それ以外は未定義）

- `dashboardReady` は、初回表示、タブ（グループ）の切り替え、パネル1枚の再読み込み（パネルの修正画面を閉じたとき）のたびに発生します。
- パネル1枚の再読み込みのときは `e.detail.reload` が `true` で、`e.detail.items` はそのパネル1件だけです。

2. パネルを囲むHTML構造

新版では、パネル（.card）の外側に配置ライブラリ（GridStack）用の要素が2つ増えました。

【旧版】

```
<div id="dashboard">
  <div class="card w6 h2 type_list" id="card-123">
    <div class="db_title"><h2>タイトル</h2></div>
    <div class="inner"> ... </div>
  </div>
</div>
```

【新版】

```

<div id="dashboard" class="grid-stack">
  <div class="grid-stack-item" id="widget_123" gs-x="0" gs-y="0" gs-
w="12" gs-h="6">
    <div class="grid-stack-item-content">
      <div class="card type_list" id="card-123" style="background-co
lor: #ffffff;">
        <div class="db_title"><h2>タイトル</h2></div>
        <div class="inner"> ... </div>
      </div>
    </div>
  </div>
</div>

```

- パネル本体（`#card-123` と、その中の `.db_title` `.inner` など）の構造は変わっていません。
- 大きさと位置は、`.card` のクラス（`w1` ～ `w6`、`h1` ～ `h4` など）ではなく、`.grid-stack-item` の属性（`gs-x` `gs-y` `gs-w` `gs-h`）で決まります。横幅は12列、高さは1行50pxです。
[例] 旧版の `w6` は `gs-w="12"`、`h2` は `gs-h="6"` に相当します。
- `.card` には、パネルの色がインラインの `style` として付きます。CSSで `.card` の背景色を指定しても、インラインの指定が優先されるため効きません。
- スマートフォンのような狭い画面幅では1列に折り返され、`gs-w` は属性ごと消えます（保存されている幅の値は変わりません）。そのため、`gs-w` の値で判定する処理は避けてください。

3. CSSの効き方

■ 無くなった・名前が変わった要素

`.dashboard_edit`（右上の操作エリア）

`.dashboard_toolbar`

操作エリアを隠すCSS

`.move`（パネルをドラッグアンドドロップで移動するためのハンドル）

無し（「レイアウト変更」中にパネル全体を掴む）

ハンドルを隠す・装飾するCSS

`.card` の `w1` ～ `w6`
`h1` ～ `h4`

`.grid-stack-item` の `gs-w` `gs-h` 属性

大きさをクラスで判定するCSS

■ 幅・高さ・余白が変わった要素

- `.card` の `width: 100%` は、ダッシュボード全体ではなく直接の親になった `.grid-stack-item-content` の幅に対する100%になります。この幅は `.grid-stack-item` の `gs-w` の値と同じです。
- 見出し `.db_title` の高さ本文 `.inner` の上余白は 40px → 30px に変更されています。

■ スクロールの範囲

- 縦スクロールは、ページ全体ではなく `.dashboard_scroll` の中で行うようになりました。
- `.dashboard_scroll` の高さは、旧版でパネルを表示できた高さより 20～30px ほど低くなります。パネルの合計の高さが画面の高さに近い構成では、旧版では出なかったスクロールバーが出ることがあります。

4. パネルの大きさ・並び順・位置の付与

- 旧版では、パネルは保存された並び順のとおり HTML へ並べられ、カスタムコードで HTML の順番を入れ替えれば見た目も変わりました。新版では、パネルごとに座標（x, y, 幅, 高さ）がユーザーごとに保存され、その座標の位置に配置されます。

- 位置や大きさなどを変えるには座標を書き換える必要があり、書き換えた座標は `savePanels()` で保存できます。

5. 旧版の動作に影響しないコードの書き方

新版向けの修正は、新版にしか無いものを条件にすることで旧版の動作に影響させないことができます。

- CSSは `#dashboard.grid-stack` から始めるセレクトタで書く（`.grid-stack` は新版にしかありません）
- JavaScriptは `dashboardReady` で実行する（旧版表示ではこのイベントは発生しません）
- 旧版用の処理を新版で止めたい場合は、
`document.querySelector('#dashboard.grid-stack')` が見つかるかで判定できます（`.grid-stack-item` は後から挿入されるので読み込み完了時の判定には使えません）

書き換え例

●読み込み完了を待つ処理

以下の処理は旧版を削除せず新版を追加する形で反映すると、旧・新版どちらでも動作させることが可能です。

【旧】

```
$(function () {  
    $('.wks_markdown p a').attr('target', '_self');  
});  
$(document).ajaxComplete(function () {  
    $('.wks_markdown p a').attr('target', '_self');  
});
```

【新】

```
document.addEventListener('dashboardReady', function () {  
    $('.wks_markdown p a').attr('target', '_self');  
});
```

●操作エリアを隠すCSS

【旧】

```
.dashboard_edit { display: none !important; }
```

【新】

```
.dashboard_toolbar { display: none !important; }
```

●パネルをダッシュボードの全幅に広げていたCSS

旧版のレイアウトを新版でも使用したい場合、次のように座標指定を打ち消すことで実現可能です。

```
#dashboard.grid-stack { height: auto !important; margin-bottom: 0; }  
#dashboard.grid-stack > .grid-stack-item { position: static !important;  
width: 100% !important; height: auto !important; }  
#dashboard.grid-stack > .grid-stack-item > .grid-stack-item-content  
{ position: static !important; height: auto !important; overflow: visible !important; }  
#dashboard.grid-stack .card { position: relative; display: flow-root; }  
#dashboard.grid-stack .card .db_title { min-height: 40px; }  
#dashboard.grid-stack .card .inner { margin-top: 40px; }
```

なお、この書き方では、新版の「レイアウト変更」で移動や大きさの変更をしても見た目には反映されなくなります。

並び順は保存されている順（旧版の並び）になり、縦スクロールの範囲は新版のままです（[3. CSSの効き方](#) 参照）。

※ この CSS はフリーテキストだけでなく、グラフや一覧などすべてのパネルに適用されます。フリーテキストのパネルはテキストの行数や画像の大きさに応じて高さが決まるため問題ありませんが、**グラフや一覧のパネルは配置用の要素の高さに合わせて中身の大きさを決めているため、この方法では高さが崩れることがあります。**

●決まった順番に並べるJavaScript

旧版で `#dashboard` の子要素を並べ替えていたコードは、新版では `dashboardReady` で `.grid-stack-item` の属性の座標を付け直す必要があります。以下のコードは横並びを保ったままパネルを並べる例です。

```
var desiredOrder = ['タイトル1', 'タイトル2', 'タイトル3'];

document.addEventListener('dashboardReady', function (e) {
    var detail = (e && e.detail) || {};
    if (detail.reload) return; // パネル1枚の再読込では動かさない
    var grid = detail.grid || window.gridInstance;
    if (!grid || !grid.engine || !grid.engine.nodes || !grid.engine.nodes.length) return;

    var nodes = grid.engine.nodes.slice();
    var byName = {};
    nodes.forEach(function (node) {
        var el = node.el.querySelector('.db_title h2');
        var name = el ? el.textContent.trim().split('\n')[0].trim() : '';
        if (name && !byName[name]) byName[name] = node;
    });
```

```

var ordered = [];
desiredOrder.forEach(function (name) {
    if (byName[name]) { ordered.push(byName[name]); delete byName[name]; }
});
nodes.filter(function (n) { return ordered.indexOf(n) < 0; })
    .sort(function (a, b) { return (a.y - b.y) || (a.x - b.x); })
    .forEach(function (n) { ordered.push(n); }); // 配列に無い
パネルは後ろに付ける

var cols = (typeof grid.getColumn === 'function') ? grid.getColumn
() : 12;
var x = 0, y = 0, rowH = 0, changed = false, layout = [];
ordered.forEach(function (node) {
    var w = Math.min(node.w || 1, cols);
    if (x + w > cols) { x = 0; y += rowH; rowH = 0; } // 12列に収ま
らなければ次の段へ
    layout.push({ node: node, x: x, y: y });
    if (node.x !== x || node.y !== y) changed = true;
    x += w;
    rowH = Math.max(rowH, node.h || 1);
});
if (!changed) return;

grid.batchUpdate();
layout.forEach(function (p) { grid.update(p.node.el, { x: p.x, y:
p.y }); });
grid.batchUpdate(false);
if (typeof savePanels === 'function') savePanels(); // 利用者ごとの
配置として保存

window.dispatchEvent(new Event('resize')); // グラフの再描
画
});

```