

コードブロックのサンプル

情報科学科

Rai

2024 年 5 月 14 日

Hello, world!

```
1 fn main() {  
2     println!("Hello, world!");  
3 }
```

Custom Callout

```
1 #import "@local/jsreport:0.1.0": callout, create-callout  
2  
3 #create-callout(  
4     "spark",  
5     (  
6         "Spark",  
7         image.decode("<svg width=\"15\" height=\"15\" viewBox=\"0 0 15 15\" fill=\"none\"  
xmlns=\"http://www.w3.org/2000/svg\"><path d=\"M8.69667 0.0403541C8.90859 0.131038 9.03106  
0.354857 8.99316 0.582235L8.0902 6.00001H12.5C12.6893 6.00001 12.8625 6.10701 12.9472  
6.27641C13.0319 6.4458 13.0136 6.6485 12.8999 6.80001L6.89997 14.8C6.76167 14.9844 6.51521  
15.0503 6.30328 14.9597C6.09135 14.869 5.96888 14.6452 6.00678 14.4178L6.90974  
9H2.49999C2.31061 9 2.13748 8.893 2.05278 8.72361C1.96809 8.55422 1.98636 8.35151 2.09999  
8.2L8.09997 0.200038C8.23828 0.0156255 8.48474 -0.0503301 8.69667 0.0403541ZM3.49999  
8.00001H7.49997C7.64695 8.00001 7.78648 8.06467 7.88148 8.17682C7.97648 8.28896 8.01733  
8.43723 7.99317 8.5822L7.33027 12.5596L11.5 7.00001H7.49997C7.353 7.00001 7.21347 6.93534  
7.11846 6.8232C7.02346 6.71105 6.98261 6.56279 7.00678 6.41781L7.66968 2.44042L3.49999  
8.00001Z\" fill=\"currentColor\" fill-rule=\"evenodd\" clip-rule=\"evenodd\"></path></svg>")  
8     )  
9 )  
10  
11 #callout("spark") [  
12     Sparking!!  
13 ]
```

remark-callout

```
1 import { defu } from "defu";  
2 import type { Properties } from "hast";  
3 import type * as mdast from "mdast";  
4 import type { Plugin } from "unified";  
5 import { visit } from "unist-util-visit";  
6 import type { VFile } from "vfile";  
7  
8 export type Options = OptionsBuilder<NodeOptions | NodeOptionsFunction>;  
9  
10 export type OptionsBuilder<N> = {  
11     /**
```

```

12     * The root node of the callout.
13     *
14     * @default
15     * (callout) => ({
16     *   tagName: callout.isFoldable ? "details" : "div",
17     *   properties: {
18     *     dataCallout: true,
19     *     dataCalloutType: callout.type,
20     *     open: callout.defaultFolded === undefined ? false : !callout.defaultFolded,
21     *   },
22     * })
23     */
24     root?: N;
25
26     /**
27     * The title node of the callout.
28     *
29     * @default
30     * (callout) => ({
31     *   tagName: callout.isFoldable ? "summary" : "div",
32     *   properties: {
33     *     dataCalloutTitle: true,
34     *   },
35     * })
36     */
37     title?: N;
38
39     /**
40     * The body node of the callout.
41     *
42     * @default
43     * () => ({
44     *   tagName: "div",
45     *   properties: {
46     *     dataCalloutBody: true,
47     *   },
48     * })
49     */
50     body?: N;
51
52     /**
53     * A list of callout types that are supported.
54     * - If `undefined`, all callout types are supported. This means that this plugin will not
55     check if the given callout type is in `callouts` and never call `onUnknownCallout`.
56     * - If a list, only the callout types in the list are supported. This means that if the
57     given callout type is not in `callouts`, this plugin will call `onUnknownCallout`.
58     * @example ["info", "warning", "danger"]
59     * @default undefined
60     */
61     callouts?: string[] | null;
62
63     /**
64     * A function that is called when the given callout type is not in `callouts`.
65     * - If the function returns `undefined`, the callout is ignored. This means that the
66     callout is rendered as a normal blockquote.

```

```

65     * - If the function returns a `Callout`, the callout is replaced with the returned
    `Callout`.
66     */
67     onUnknownCallout?: (callout: Callout, file: VFile) => Callout | undefined;
68 };
69
70 export type NodeOptions = {
71     /**
72      * The HTML tag name of the node.
73      *
74      * @see https://github.com/syntax-tree/hast?tab=readme-ov-file#element
75      */
76     tagName: string;
77
78     /**
79      * The html properties of the node.
80      *
81      * @see https://github.com/syntax-tree/hast?tab=readme-ov-file#properties
82      * @see https://github.com/syntax-tree/hast?tab=readme-ov-file#element
83      * @example { "className": "callout callout-info" }
84      */
85     properties: Properties;
86 };
87
88 export type NodeOptionsFunction = (callout: Callout) => NodeOptions;
89
90 export const defaultOptions: Required<Options> = {
91     root: (callout) => ({
92         tagName: callout.isFoldable ? "details" : "div",
93         properties: {
94             dataCallout: true,
95             dataCalloutType: callout.type,
96             open:
97                 callout.defaultFolded === undefined ? false : !callout.defaultFolded,
98         },
99     }),
100     title: (callout) => ({
101         tagName: callout.isFoldable ? "summary" : "div",
102         properties: {
103             dataCalloutTitle: true,
104         },
105     }),
106     body: () => ({
107         tagName: "div",
108         properties: {
109             dataCalloutBody: true,
110         },
111     }),
112     callouts: null,
113     onUnknownCallout: () => undefined,
114 };
115
116 const initOptions = (options?: Options) => {
117     const defaultedOptions = defu(options, defaultOptions);
118
119     return Object.fromEntries(
120         Object.entries(defaultedOptions).map(([key, value]) => {

```

```

121     if (
122         ["root", "title", "body"].includes(key) &&
123         typeof value !== "function"
124     )
125         return [key, () => value];
126
127     return [key, value];
128 },
129 ) as Required<OptionsBuilder<NodeOptionsFunction>>;
130 };
131
132 /**
133  * A remark plugin to parse callout syntax.
134  */
135 export const remarkCallout: Plugin<[Options?], mdast.Root> = (_options) => {
136     const options = initOptions(_options);
137
138     return (tree, file) => {
139         visit(tree, "blockquote", (node) => {
140             const paragraphNode = node.children[0];
141             if (paragraphNode.type !== "paragraph") return;
142
143             const calloutTypeTextNode = paragraphNode.children[0];
144             if (calloutTypeTextNode.type !== "text") return;
145
146             // Parse callout syntax
147             // e.g. "[!note] title"
148             const [calloutTypeText, ...calloutBodyText] =
149                 calloutTypeTextNode.value.split("\n");
150             const calloutData = parseCallout(calloutTypeText);
151             if (calloutData == null) return;
152             if (
153                 options.callouts !== null &&
154                 !options.callouts.includes(calloutData.type)
155             ) {
156                 const newCallout = options.onUnknownCallout(calloutData, file);
157                 if (newCallout == null) return;
158
159                 calloutData.type = newCallout.type;
160                 calloutData.isFoldable = newCallout.isFoldable;
161                 calloutData.title = newCallout.title;
162             }
163
164             // Generate callout root node
165             node.data = {
166                 ...node.data,
167                 hName: options.root(calloutData).tagName,
168                 hProperties: {
169                     // @ts-ignore error TS2339: Property 'hProperties' does not exist on type
170                     'BlockquoteData':
171                         ...node.data?.hProperties,
172                         ...options.root(calloutData).properties,
173                 },
174             };
175
176             // Generate callout body node
177             const bodyNode: (mdast.BlockContent | mdast.DefinitionContent)[] = [

```

```

177     {
178         type: "paragraph",
179         children: [],
180     },
181     ...node.children.splice(1),
182 ];
183 if (bodyNode[0].type !== "paragraph") return; // type check
184 if (calloutBodyText.length > 0) {
185     bodyNode[0].children.push({
186         type: "text",
187         value: calloutBodyText.join("\n"),
188     });
189 }
190
191 // Generate callout title node
192 const titleNode: mdast.Paragraph = {
193     type: "paragraph",
194     data: {
195         hName: options.title(calloutData).tagName,
196         hProperties: {
197             ...options.title(calloutData).properties,
198         },
199     },
200     children: [],
201 };
202 if (calloutData.title !== null) {
203     titleNode.children.push({
204         type: "text",
205         value: calloutData.title,
206     });
207 }
208 if (calloutBodyText.length ≤ 0) {
209     for (const [i, child] of paragraphNode.children.slice(1).entries()) {
210         // All inline node before the line break is added as callout title
211         if (child.type !== "text") {
212             titleNode.children.push(child);
213             continue;
214         }
215
216         // Add the part before the line break as callout title and the part after as
217         callout body
218         const [titleText, ...bodyTextLines] = child.value.split("\n");
219         if (titleText) {
220             // Add the part before the line break as callout title
221             titleNode.children.push({
222                 type: "text",
223                 value: titleText,
224             });
225         }
226         if (bodyTextLines.length > 0) {
227             // Add the part after the line break as callout body
228             if (bodyNode[0].type !== "paragraph") return;
229             bodyNode[0].children.push({
230                 type: "text",
231                 value: bodyTextLines.join("\n"),
232             });
233             // Add all nodes after the current node as callout body

```

```

233         bodyNode[0].children.push(...paragraphNode.children.slice(i + 2));
234         break;
235     }
236 }
237 } else {
238     // Add all nodes after the current node as callout body
239     bodyNode[0].children.push(...paragraphNode.children.slice(1));
240 }
241
242 // Add body and title to callout root node children
243 node.children = [
244     titleNode,
245     {
246         type: "blockquote",
247         data: {
248             hName: options.body(calloutData).tagName,
249             hProperties: {
250                 ...options.body(calloutData).properties,
251             },
252         },
253         children: bodyNode,
254     },
255 ];
256 });
257 };
258 };
259
260 export type Callout = {
261     /**
262      * The type of the callout.
263      */
264     type: string;
265
266     /**
267      * Whether the callout is foldable.
268      */
269     isFoldable: boolean;
270
271     /**
272      * Whether the callout is folded by default.
273      */
274     defaultFolded?: boolean;
275
276     /**
277      * The title of the callout.
278      */
279     title?: string;
280 };
281
282 /**
283  * @example
284  * ```
285  * const callout = parseCallout("[!info]"); // { type: "info", isFoldable: false, title:
undefined }
286  * const callout = parseCallout("[!info]"); // undefined
287  * ```
288  */

```

```

289 export const parseCallout = (
290   text: string | null | undefined,
291 ): Callout | undefined => {
292   if (text === null) return;
293
294   const match = text.match(
295     /^\[!(?<type>.+?)\](?<isFoldable>[-+]?)\s?(?<title>.+)?$/,
296   );
297   if (match?.groups?.type === null) return undefined;
298
299   return {
300     type: match.groups.type,
301     isFoldable: match.groups.isFoldable !== null,
302     defaultFolded:
303       match.groups.isFoldable === null
304         ? undefined
305         : match.groups.isFoldable === "-" ? true : false,
306     title: match.groups.title,
307   };
308 };

```
