转换

目录

- Transform direction: to and from
- 转换类型
  - Block 类型
  - 输入
  - Files
  - 前缀转换
  - 原始类型
    - 模式与内容模型
  - 短代码
- ungroup blocks

区块转换 API 允许区块与其他区块之间进行相互转换，也能从其他实体转换而来。支持此 API 的现有实体包括短代码、文件、正则表达式和原始 DOM 节点。

Transform direction: `to` and `from`

A block declares which transformations it supports via the optional `transforms` key of the block configuration, whose subkeys `to` and `from` hold an array of available transforms for every direction. Example:

```
export const settings = {
	title: 'My Block Title',
	description: 'My block description',
	/* ... */
	transforms: {
		from: [
			/* supported from transforms */
		],
		to: [
```

```
□□□/* supported to transforms */
□□],
□},
};
```

转换类型

本节介绍现有转换块支持的类型：

- block
- enter
- files
- prefix
- raw
- shortcode

Block 类型

此类转换支持 从 和 到 两个方向，允许将块转换为不同的块。它在块工具栏中有相应的 UI 控件。

`block` 类型的转换是一个对象，包含以下参数：

- **type** *(字符串)*：值为 `block`。
- **blocks** *(数组)*：已知块类型的列表。它也接受通配符值 (`"*"`)，表示该转换适用于 所有块类型（例如：所有块都可以转换为 `core/group`）。
- **transform** *(函数)*：
  一个回调函数，接收被处理块的属性和内部块。它应返回一个块对象或块对象数组。
- **isMatch** *(函数，可选)*：
  一个回调函数，第一个参数接收块属性，第二个参数接收块对象，并应返回一个布尔值。从此函数返回 `false` 将阻止该转换可用，并且不会作为选项显示给用户。
- **isMultiBlock** *(布尔值，可选)*：当选择多个块时是否可以应用此转换。如果为 true，`transform` 函数的第一个参数将是一个包含每个选定块属性的数组，第二个参数是每个选定块的内部块数组。默认为 false。
- **priority** *(数字，可选)*：控制转换应用的优先级，较低的值将比较高的值优先。这很像 WordPress 钩子。与钩子类似，未设置时默认优先级为 `10`。

示例：从段落块转换为标题块

要声明此转换，我们将以下代码添加到标题块配置中，该配置使用了 `wp-blocks` 包中的 `createBlock` 函数。

```
transforms: {
    from: [
        {
            type: 'block',
            blocks: [ 'core/paragraph' ],
            transform: ( { content } ) => {
                return createBlock( 'core/heading', {
                    content,
                } );
            },
        },
    ]
},
```

示例：具有 **InnerBlocks** 的块

具有 InnerBlocks 的块也可以与另一个具有 InnerBlocks 的块相互转换。

```
transforms: {
    to: [
        {
            type: 'block',
            blocks: [ 'some/block-with-innerblocks' ],
            transform: ( attributes, innerBlocks ) => {
                return createBlock(
                    'some/other-block-with-innerblocks',
                    attributes,
                    innerBlocks
                );
            },
        },
    ],
},
```

输入

此类转换支持 *from* 方向，允许根据用户输入的内容创建区块。当用户输入某些内容并按下 ENTER 键后，转换将在新的区块行中应用。

`enter` 类型的转换是一个包含以下参数的对象：

- **type** *(字符串)*: 值为 `enter`
- **regExp** *(正则表达式)*: 用作匹配器的正则表达式。若值匹配，则应用该转换
- **transform** *(函数)*: 接收包含已输入值的 `content` 字段对象的回调函数。应返回一个区块对象或区块对象数组
- **priority** *(数字，可选)*: 控制转换应用的优先级，数值越低优先级越高。其行为类似于 WordPress 钩子。与钩子类似，未设置时默认优先级为 `10`

示例：从 `---` 到分隔符区块

当用户输入三个连字符后按下 ENTER 键时创建分隔符区块，可使用以下代码：

```
transforms = {
□from: [
□□{
□□□type: 'enter',
□□□regExp: /^-{3,}$/,
□□□transform: () => createBlock( 'core/separator' ),
□□},
□],
};
```

Files

This type of transformations support the *from* direction, allowing blocks to be created from files dropped into the editor.

A transformation of type `files` is an object that takes the following parameters:

- **type** *(string)*: the value `files`.
- **transform** *(function)*: a callback that receives the array of files being processed. It should return a block object or an array of block objects.
- **isMatch** *(function, optional)*: a callback that receives the array of files being processed and should return a boolean. Returning `false` from this function will prevent the transform from being applied.
- **priority** *(number, optional)*: controls the priority with which a transform is applied, where a lower value will take precedence over higher values. This behaves much like a WordPress hook. Like hooks, the default priority is `10` when not otherwise set.

Example: from file to File block

To create a File block when the user drops a file into the editor we can use the following code:

```
transforms: {
	from: [
		{
			type: 'files',
			isMatch: ( files ) => files.length === 1,
			// By defining a lower priority than the default of 10,
			// we make that the File block to be created as a fallback,
			// if no other transform is found.
			priority: 15,
			transform: ( files ) => {
				const file = files[ 0 ];
				const blobURL = createBlobURL( file );
				// File will be uploaded in componentDidMount()
				return createBlock( 'core/file', {
					href: blobURL,
					fileName: file.name,
					textLinkHref: blobURL,
				} );
			},
		},
	];
}
```

前缀转换

此类转换支持 *from* 方向，允许根据用户输入的文本创建区块。当用户在新行输入文本并添加尾随空格时，将应用这些转换。

`prefix` 类型的转换是一个包含以下参数的对象：

- **type** *(字符串)*：值为 `prefix`。
- **prefix** *(字符串)*：匹配此转换的字符或字符序列。
- **transform** *(函数)*：接收输入内容的回调函数。它应返回一个区块对象或区块对象数组。
- **priority** *(数字，可选)*：控制转换应用的优先级，数值越低优先级越高。其行为类似于 WordPress 钩子。与钩子类似，未设置时默认优先级为 `10`。

示例：从文本到自定义区块

如果我们想在用户输入问号时创建自定义区块，可以使用以下代码：

```
transforms: {
□from: [
□□{
□□□type: 'prefix',
□□□prefix: '?',
□□□transform( content ) {
□□□□return createBlock( 'my-plugin/question', {
□□□□□content,
□□□□} );
□□□},
□□},
□];
}
```

原始类型

此类转换支持 *from* 方向，允许从原始 HTML 节点创建区块。当用户从区块设置 UI 菜单中执行 "转换为区块 " 操作，以及某些内容被粘贴或拖放到编辑器中时，这些转换会被应用。

`raw` 类型的转换是一个对象，包含以下参数：

- **type** *(*字符串 *)*：值为 `raw` 。
- **transform** *(*函数，可选 *)*：
  接收正在处理的节点的回调函数。它应返回一个区块对象或区块对象数组。
- **schema** *(*对象 |函数，可选 *)*：定义一个用于检测和处理粘贴内容的 HTML 内容模型 。参见 下文。
- **selector** *(*字符串，可选 *)*：一个 CSS 选择器字符串，用于根据 element.matches 方法判断元素是否匹配。如果元素不匹配，则不会执行转换。这是使用 `isMatch` 的简写和替代方案，如果同时存在， `isMatch` 将优先。
- **isMatch** *(*函数，可选 *)*：接收正在处理的节点的回调函数，应返回一个布尔值。从此函数返回 `false` 将阻止应用转换。
- **priority** *(*数字，可选 *)*：控制应用转换的优先级，数值越低优先级越高。其行为类似于 WordPress 钩子。与钩子类似，未设置时默认优先级为 `10` 。

示例：从 **URL** 到嵌入区块

如果我们想在用户向编辑器中粘贴某个 URL 时创建一个嵌入区块，可以使用以下代码：

```
transforms: {
    from: [
        {
            type: 'raw',
```

```
            isMatch: ( node ) =>
                node.nodeName === 'P' &&
                /^\s*(https?:\/\/\S+)\s*$/i.test( node.textContent ),
            transform: ( node ) => {
                return createBlock( 'core/embed', {
                    url: node.textContent.trim(),
                } );
            },
        },
    ],
}
```

模式与内容模型

粘贴内容时，可以定义一个用于验证和处理粘贴内容的 内容模型。粘贴到编辑器中的 HTML 通常包含 应该转移的元素和 不应该 转移的元素的混合。例如，考虑将 `<span class="time">12:04 pm</span>` 粘贴到编辑器中。我们希望复制 `12:04 pm` 并省略 `<span>` 及其 `class` 属性，因为这些元素和属性在复制后不会保留其原有的含义或结构。

在编写 `raw` 转换器时，你可以通过提供 `schema` 来控制此行为，该模式描述了允许的内容，并将在尝试与你的区块匹配之前应用于清理粘贴的内容。这些模式会传递给 `@wordpress/dom` 中的 `cleanNodeList`；请查看那里以获取 模式的完整描述。

```
schema = { span: { children: { '#text': {} } } };
```

示例：自定义内容模型

假设我们想要匹配以下 HTML 片段并将其转换为某种自定义文章预览区块。

```
<div data-post-id="13">
□<h2>The Post Title</h2>
□<p>Some <em>great</em> content.</p>
</div>
```

我们想要告诉编辑器允许内部的 `h2` 和 `p` 元素。我们通过提供以下模式来实现这一点。在这个例子中，我们使用了函数形式，它接受一个提供 `phrasingContentSchema` 的参数（以及一个布尔值 `isPaste`，指示转换操作是否从粘贴文本开始）。`phrasingContentSchema` 是预定义的，用于匹配 HTML 短语元素，例如 `<strong>`、`<sup>` 和 `<kbd>`。任何我们期望使用 `<RichText />` 组件的地方都是允许短语内容的好地方，否则我们将在转换时丢失所有文本格式。

```
schema = ({ phrasingContentSchema }) => {
    div: {
        required: true,
        attributes: [ 'data-post-id' ],
        children: {
            h2: { children: phrasingContentSchema },
            p: { children: phrasingContentSchema }
        }
    }
}
```

当我们成功匹配此内容时，除了 `data-post-id` 之外的所有 HTML 属性都将被剥离，并且如果我们在给定的 `div` 内部有其他 HTML 排列，那么它将不会匹配我们的转换器。同样，如果我们在其中找到 `<h3>` 而不是 `<h2>`，我们也会匹配失败。

当想要匹配包含非短语内容的 HTML 片段时，例如带有 `<summary>` 的 `<details>`，模式最为重要。如果不声明自定义模式，编辑器将在尝试通过任何区块转换器运行这些其他结构之前跳过它们。

短代码

此类转换支持 *from* 方向，允许从短代码创建区块。它作为 `raw` 转换过程的一部分应用。

`shortcode` 类型的转换是一个包含以下参数的对象：

- **type** *(*字符串 *)*：值为 `shortcode`。
- **tag** *(*字符串 | 数组 *)*：此转换可处理的短代码标签或短代码别名列表。
- **transform** *(*函数，可选 *)*：一个回调函数，第一个参数接收短代码属性，第二个参数接收 WPShortcodeMatch。它应返回一个区块对象或区块对象数组。定义此参数时，它将优先于 `attributes` 参数。
- **attributes** *(*对象，可选 *)*：根据 区块配置对象 定义的属性结构，表示区块属性应从何处获取的对象。如果某个特定属性包含 `shortcode` 键，它应是一个函数，第一个参数接收短代码属性，第二个参数接收 WPShortcodeMatch，并返回将存储在区块注释中的属性值。
- **isMatch** *(*函数，可选 *)*：一个回调函数，接收根据 Shortcode API 获取的短代码属性，并应返回一个布尔值。从此函数返回 `false` 将阻止短代码转换为此区块。
- **priority** *(*数字，可选 *)*：控制转换应用的优先级，较低值将优先于较高值。其行为类似于 WordPress 钩子。与钩子类似，未设置时默认优先级为 `10`。

示例：使用 `transform` 从短代码到区块

可以使用 `transform` 方法将现有短代码转换为其对应的区块。

```
transforms: {
    from: [
        {
            type: 'shortcode',
            tag: 'video',
            transform( { named: { src } } ) {
                return createBlock( 'core/video', { src } );
            },
            // 当短代码没有正确的 ID 时，
            // 阻止其转换为此区块。
            isMatch( { named: { id } } ) {
                return id === 'my-id';
            },
        },
    ],
},
```

示例：使用 **`attributes`** 从短代码到区块

可以使用 `attributes` 参数将现有短代码转换为其对应的区块。

```
transforms: {
    from: [
        {
            type: 'shortcode',
            tag: 'youtube',
            attributes: {
                url: {
                    type: 'string',
                    source: 'attribute',
                    attribute: 'src',
                    selector: 'img',
                },
                align: {
                    type: 'string',
                    // The shortcode function will extract
                    // the shortcode atts into a value
                    // to be sourced in the block's comment.
```

```
                    shortcode: ( { named: { align = 'alignnone' } } ) => {
                        return align.replace( 'align', '' );
                    },
                },
            },
            // Prevent the shortcode to be converted
            // into this block when it doesn't
            // have the proper ID.
            isMatch( { named: { id } } ) {
                return id === 'my-id';
            },
        },
    ]
},
```

`ungroup` blocks

Via the optional `transforms` key of the block configuration, blocks can use the `ungroup` subkey to define the blocks that will replace the block being processed. These new blocks will usually be a subset of the existing inner blocks, but could also include new blocks.

If a block has an `ungroup` transform, it is eligible for ungrouping, without the requirement of being the default grouping block. The UI used to ungroup a block with this API is the same as the one used for the default grouping block. In order for the Ungroup button to be displayed, we must have a single grouping block selected, which also contains some inner blocks.

ungroup is a callback function that receives the attributes and inner blocks of the block being processed. It should return an array of block objects.

Example:

```
export const settings = {
	title: 'My grouping Block Title',
	description: 'My grouping block description',
	/* ... */
	transforms: {
		ungroup: ( attributes, innerBlocks ) =>
			innerBlocks.flatMap( ( innerBlock ) => innerBlock.innerBlocks ),
	},
};
```

版本号 #2

由 壳壳虫 创建于 2026-03-26 17:08:43 UTC

由 壳壳虫 更新于 2026-03-26 17:08:43 UTC