Attributes

- [redacted]
- [redacted]
- [redacted]
  - attribute [redacted]
  - text source
  - html [redacted]
  - query [redacted]
  - [redacted]
    - [redacted]
- [redacted]
- [redacted]

Block attributes provide information about the data stored by a block. For example, rich content, a list of image URLs, a background color, or a button title.

A block can contain any number of attributes, and these are specified by the `attributes` field - an object where each key is the name of the attribute, and the value is the attribute definition.

The attribute definition will contain, at a minimum, either a `type` or an `enum`. There may be additional fields.

Example: Attributes object defining three attributes - `url`, `title`, and `size`.

```
{
	url: {
		type: 'string',
		source: 'attribute',
		selector: 'img',
		attribute: 'src',
	},
	title: {
		type: 'string',
```

```
},
size: {
enum: [ 'large', 'small' ],
},
}
```

When a block is parsed this definition will be used to extract data from the block content. Anything that matches will be available to your block through the `attributes` prop.

This parsing process can be summarized as:

1. Extract value from the `source`.
2. Check value matches the `type`, or is one of the `enum` values.

Example: Attributes available in the `edit` and function, using the above attributes definition.

```
function YourBlockEdit( { attributes } ) {
return (
<p>URL is { attributes.url }, title is { attributes.title }, and size is { attributes.size
}.</p>
)
}
```

The block is responsible for using the `save` function to ensure that all attributes with a `source` field are saved according to the attributes definition. This is not automatic.

Attributes without a `source` will be automatically saved in the block comment delimiter.

For example, using the above attributes definition you would need to ensure that your `save` function has a corresponding img tag for the `url` attribute. The `title` and `size` attributes will be saved in the comment delimiter.

Example: Example `save` function that contains the `url` attribute.

```
function YourBlockSave( { attributes } ) {
return (
<img src={ attributes.url } />
)
}
```

The saved HTML will contain the `title` and `size` in the comment delimiter, and the `url` in the `img` node.

```
<!-- block:your-block {"title":"hello world","size":"large"} -->
<img src="/image.jpg" />
<!-- /block:your-block -->
```

If an attribute changes over time then a block deprecation can help migrate from an older attribute, or remove it entirely.

类型验证

`type` 表示属性存储的数据类型。它不表示数据存储的位置，该信息由 `source` 字段定义。

除非提供了 `enum`，否则 `type` 是必需的。 `type` 可以与 `enum` 一起使用。

`type` 字段必须是以下之一：

- `null`
- `boolean`
- `object`
- `array`
- `string`
- `integer`
- `number`（与 `integer` 相同）

请注意， `object` 的有效性由您的 `source` 决定。有关示例，请参阅下面的 `query` 详细信息。

枚举验证

属性可以被定义为固定值集合中的一个值。这通过 `enum` 来指定，它包含一个允许值的数组：

示例： `enum` 示例。

```
{
□size: {
□□enum: [ 'large', 'small', 'tiny' ]
□}
}
```

值来源

属性来源用于定义如何从已保存的文章内容中提取属性值。它们提供了一种机制，将已保存的标记映射到块的 JavaScript 表示形式。

可用的 `source` 值包括：

- `（无值）` - 当未指定 `source` 时，数据存储在块的 注释分隔符 中。
- `attribute` - 数据存储在 HTML 元素属性中。
- `text` - 数据存储在 HTML 文本中。
- `html` - 数据存储在 HTML 中。这通常由 `RichText` 使用。
- `query` - 数据存储为对象数组。
- `meta` - 数据存储在文章元数据中（已弃用）。

`source` 字段通常与 `selector` 字段结合使用。如果未指定选择器参数，则源定义针对块的根节点运行。如果指定了选择器参数，它将针对块内匹配的元素运行。

`selector` 可以是 HTML 标签，或任何可通过 querySelector 查询的内容，例如类或 id 属性。下面给出了示例。

例如，`img` 的 `selector` 将匹配 `img` 元素，而 `img.class` 将匹配具有 `class` 类的 `img` 元素。

在底层，属性来源是 hpq 提供的功能的超集，这是一个用于将 HTML 标记解析和查询为对象形状的小型库。

总结来说，`source` 决定了数据在内容中的存储位置，而 `type` 决定了该数据是什么。为了减少存储的数据量，通常最好将尽可能多的数据存储在 HTML 中，而不是作为注释分隔符内的属性。

`attribute` 来源

使用 `attribute` 来源可从标记中的属性提取值。该属性由 `attribute` 字段指定，且必须提供。

示例：从块标记中的图像提取 `src` 属性。

保存的内容：

```
<div>
块内容

<img src="https://lorempixel.com/1200/800/" />
</div>
```

属性定义:

```
{
  url: {
    type: 'string',
    source: 'attribute',
    selector: 'img',
    attribute: 'src',
  }
}
```

块中可用的属性:

```
{ "url": "https://lorempixel.com/1200/800/" }
```

大多数来自标记的属性都是 `string` 类型。 HTML 中的数字属性仍以字符串形式存储，不会自动转换。

示例：从块标记中的图像提取 `width` 属性。

保存的内容:

```
<div>
块内容

<img src="https://lorempixel.com/1200/800/" width="50" />
</div>
```

属性定义:

```
{
  width: {
    type: 'string',
    source: 'attribute',
    selector: 'img',
    attribute: 'width',
  }
}
```

块中可用的属性:

```
{ "width": "50" }
```

唯一的例外是检查属性是否存在时（例如，`button` 上的 `disabled` 属性）。在这种情况下，可以使用 `boolean` 类型，存储的值将是布尔值。

示例：从块标记中的按钮提取 `disabled` 属性。

保存的内容：

```
<div>
块内容

<button type="button" disabled>按钮</button>
</div>
```

属性定义：

```
{
	disabled: {
		type: 'boolean',
		source: 'attribute',
		selector: 'button',
		attribute: 'disabled',
	}
}
```

块中可用的属性：

```
{ "disabled": true }
```

`text` source

Use `text` to extract the inner text from markup. Note that HTML is returned according to the rules of `textContent`.

Example: Extract the `content` attribute from a figcaption element found in the block's markup.

Saved content:

```
<figure>
	<img src="/image.jpg" />

	<figcaption>The inner text of the figcaption element</figcaption>
```

```
</figure>
```

Attribute definition:

```
{
	content: {
		type: 'string',
		source: 'text',
		selector: 'figcaption',
	}
}
```

Attribute available in the block:

```
{ "content": "The inner text of the figcaption element" }
```

Another example, using `text` as the source, and using `.my-content` class as the selector to extract text:

Example: Extract the `content` attribute from an element with `.my-content` class found in the block's markup.

Saved content:

```
<div>
	<img src="/image.jpg" />

	<p class="my-content">The inner text of .my-content class</p>
</div>
```

Attribute definition:

```
{
	content: {
		type: 'string',
		source: 'text',
		selector: '.my-content',
	}
}
```

Attribute available in the block:

```
{ "content": "The inner text of .my-content class" }
```

`html` 源

使用 `html` 从标记中提取内部 HTML。请注意，文本的返回遵循 `innerHTML` 的规则。

示例：从块标记中找到的 figcaption 元素中提取 `content` 属性。

保存的内容：

```
<figure>
	<img src="/image.jpg" />

	<figcaption>The inner text of the <strong>figcaption</strong> element</figcaption>
</figure>
```

属性定义：

```
{
	content: {
		type: 'string',
		source: 'html',
		selector: 'figcaption',
	}
}
```

块中可用的属性：

```
{ "content": "The inner text of the <strong>figcaption</strong> element" }
```

`query` 来源

使用 `query` 从标记中提取值数组。数组的条目由 `selector` 参数决定，块内每个匹配的元素都将对应一个条目，其结构由第二个参数（属性来源对象）定义。

`query` 字段本质上是一个嵌套的块属性定义。可以（尽管不一定推荐）进一步嵌套。

示例：从块标记中的每个图像元素提取 `src` 和 `alt`。

保存的内容：

```
<div>
  <img src="https://lorempixel.com/1200/800/" alt="large image" />
  <img src="https://lorempixel.com/50/50/" alt="small image" />
</div>
```

□□□□□

```
{
  images: {
    type: 'array',
    source: 'query',
    selector: 'img',
    query: {
      url: {
        type: 'string',
        source: 'attribute',
        attribute: 'src',
      },
      alt: {
        type: 'string',
        source: 'attribute',
        attribute: 'alt',
      },
    }
  }
}
```

□□□□□□□□

```
{
  "images": [
    { "url": "https://lorempixel.com/1200/800/", "alt": "large image" },
    { "url": "https://lorempixel.com/50/50/", "alt": "small image" }
  ]
}
```

□□□□□□□□

>

> 尽管属性可以从文章的元数据中获取，但元数据属性源已被视为弃用；应改用 EntityProvider 及相关钩子 API，如 创建元数据块指南 所示。

属性可以从文章的元数据中获取，而不是从已保存文章内容中的块表示中获取。为此，需要在 `meta` 键下指定其对应元数据键的属性。

属性定义：

```
{
	author: {
		type: 'string',
		source: 'meta',
		meta: 'author'
	},
},
```

此后，块可以使用与任何属性相同的接口来读取和写入元数据属性：

```
edit( { attributes, setAttributes } ) {
	function onChange( event ) {
		setAttributes( { author: event.target.value } );
	}

	return <input value={ attributes.author } onChange={ onChange } type="text" />;
},
```

注意事项

默认情况下，元字段会被排除在文章对象的元数据之外。可以通过显式设置字段可见性来规避此限制：

```
function gutenberg_my_block_init() {
	register_post_meta( 'post', 'author', array(
		'show_in_rest' => true,
	) );
}
add_action( 'init', 'gutenberg_my_block_init' );
```

此外，请注意 WordPress 默认行为是：

- 不将元数据视为唯一值，而是返回值的数组；
- 将该数据视为字符串。

若需改变上述任一行为，可在同一 `register_post_meta` 调用中补充 `single` 和 /或 `type` 参数:

```
function gutenberg_my_block_init() {
	register_post_meta( 'post', 'author_count', array(
		'show_in_rest' => true,
		'single' => true,
		'type' => 'integer',
	) );
}
add_action( 'init', 'gutenberg_my_block_init' );
```

若要在属性中使用对象或数组，可注册 `string` 类型的属性，并使用 JSON 作为中间格式。在保存前将结构化数据序列化为 JSON，然后在服务端反序列化 JSON 字符串。请注意您需对数据完整性负责；务必进行适当的数据清理、处理缺失数据等操作。

最后，请确保在设置属性时遵循数据类型，因为框架不会自动执行元数据的类型转换。块属性中的类型错误将导致文章在保存后仍被标记为未保存状态（参见 `isEditedPostDirty`、`hasEditedAttributes`）。例如，若 `authorCount` 是整数类型，请注意事件处理器可能传递不同类型的数据，因此应显式转换值:

```
function onChange( event ) {
	props.setAttributes( { authorCount: Number( event.target.value ) } );
}
```

默认值

块属性可以包含一个默认值，当 `type` 和 `source` 与块内容中的任何内容都不匹配时，将使用此默认值。

该值由 `default` 字段提供，并且该值应符合属性的预期格式。

示例： `default` 值的示例。

```
{
	type: 'string',
	default: 'hello world'
}
```

```
{
	type: 'array',
	default: [
		{ "url": "https://lorempixel.com/1200/800/", "alt": "large image" },
```

```
    { "url": "https://lorempixel.com/50/50/", "alt": "small image" }
]
}
```

```
{
type: 'object',
default: {
width: 100,
title: 'title'
}
}
```


`role`

 `content` `content`
 `local` `local`

 `content`

```
{
    content: {
        type: 'string',
        source: 'html',
        selector: 'p',
        role: 'content',
    }
}
```

 `local`

```
{
    blob: {
        type: 'string',
        role: 'local',
    }
```

```
}
```

了解更多信息，请参阅 WordPress 6.7 开发说明。

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