编辑与保存

目录

- 编辑
  - 区块包装器属性
  - 属性
  - isSelected
  - setAttributes
- Save
  - 区块包装器属性
  - 属性
  - innerBlocks
- Examples
  - Saving Attributes to Child Elements
  - 通过序列化保存属性
- 验证
  - 验证常见问题

在客户端使用 JavaScript 注册区块时，`edit` 和 `save` 函数定义了区块在编辑器中的渲染方式、操作逻辑以及保存机制。

编辑

`edit` 函数描述了你的区块在编辑器环境中的结构。这代表了当区块被使用时，编辑器将渲染的内容。

```
import { useBlockProps } from '@wordpress/block-editor';

// ...
const blockSettings = {
	apiVersion: 3,

	// ...
```

```
edit: () => {
  const blockProps = useBlockProps();

  return <div { ...blockProps }>区块。</div>;
},
};
```

区块包装器属性

首先需要注意的是区块包装元素上使用的 `useBlockProps` React 钩子。在上面的示例中，区块包装器在编辑器中渲染了一个 "div"，但为了让古腾堡编辑器知道如何操作区块、添加区块所需的任何额外类名……区块包装元素应当应用从 `useBlockProps` React 钩子调用中获取的属性。区块包装元素应当是一个原生 DOM 元素，例如 `<div>` 和 `<table>`，或者是一个将任何额外属性传递给原生 DOM 元素的 React 组件。例如，使用 `<Fragment>` 或 `<ServerSideRender>` 组件将是无效的。

如果元素包装器需要任何额外的自定义 HTML 属性，这些属性需要作为参数传递给 `useBlockProps` 钩子。例如，要向包装器添加 `my-random-classname` 类名，可以使用以下代码：

```
import { useBlockProps } from '@wordpress/block-editor';

// ...
const blockSettings = {
  apiVersion: 3,

  // ...

  edit: () => {
    const blockProps = useBlockProps( {
      className: 'my-random-classname',
    } );

    return <div { ...blockProps }>区块。</div>;
  },
};
```

属性

`edit` 函数还会通过一个对象参数接收若干属性。您可以使用这些属性来调整区块的行为。

`attributes` 属性会展示所有可用的属性及其对应的值，这些属性在区块类型注册时通过 `attributes` 属性定义。关于如何指定属性来源，请参阅 属性文档_____ 。

在此示例中，假设我们在区块注册时定义了一个 `content` 属性，我们将在编辑函数中接收并使用该值：

```
edit: ( { attributes } ) => {
	const blockProps = useBlockProps();

	return <div { ...blockProps }>{ attributes.content }</div>;
};
```

当在编辑器中插入区块时，`attributes.content` 的值将显示在 `div` 内部。

isSelected

The isSelected property is a boolean that communicates whether the block is currently selected.

```
edit: ( { attributes, isSelected } ) => {
	const blockProps = useBlockProps();

	return (
		<div { ...blockProps }>
			Your block.
			{ isSelected && (
				<span>Shows only when the block is selected.</span>
			) }
		</div>
	);
};
```

setAttributes

This function allows the block to update individual attributes based on user interactions.

```
edit: ( { attributes, setAttributes, isSelected } ) => {
	const blockProps = useBlockProps();

	// Simplify access to attributes
	const { content, mySetting } = attributes;
```

```
	// Toggle a setting when the user clicks the button
	const toggleSetting = () => setAttributes( { mySetting: ! mySetting } );
	return (
		<div { ...blockProps }>
			{ content }
			{ isSelected && (
				<button onClick={ toggleSetting }>Toggle setting</button>
			) }
		</div>
	);
};
```

When using attributes that are objects or arrays it's a good idea to copy or clone the attribute prior to updating it:

```
// Good - a new array is created from the old list attribute and a new list item:
const { list } = attributes;
const addListItem = ( newListItem ) =>
	setAttributes( { list: [ ...list, newListItem ] } );

// Bad - the list from the existing attribute is modified directly to add the new list item:
const { list } = attributes;
const addListItem = ( newListItem ) => {
	list.push( newListItem );
	setAttributes( { list } );
};
```

Why do this? In JavaScript, arrays and objects are passed by reference, so this practice ensures changes won't affect other code that might hold references to the same data. Furthermore, the Gutenberg project follows the philosophy of the Redux library that state should be immutable—data should not be changed directly, but instead a new version of the data created containing the changes.

The `setAttribute` also supports an updater function as an argument. It must be a pure function, which takes current attributes as its only argument and returns updated attributes. This method is helpful when you want to update an value based on a previous state or when working with objects and arrays.

Note: *Since WordPress 6.9.*

```
// Toggle a setting when the user clicks the button.
const toggleSetting = () =>
	setAttributes( ( currentAttr ) => ( {
		mySetting: ! currentAttr.mySetting,
	} ) );

// Add item to the list.
const addListItem = ( newListItem ) =>
	setAttributes( ( currentAttr ) => ( {
		list: [ ...currentAttr.list, newListItem ],
	} ) );
```

Save

`save` 函数定义了如何将不同属性组合成最终标记，然后序列化到 `post_content` 中。

```
save: () => {
	const blockProps = useBlockProps.save();

	return <div { ...blockProps }>你的区块内容 </div>;
};
```

对于大多数区块， `save` 的返回值应该是一个 WordPress 元素实例，代表区块在前端网站上的显示方式。

注意： 虽然可以从 `save` 返回字符串值，但它 会被转义 。如果字符串包含 HTML 标记，标记将在网站前端按字面显示，而不是作为等效的 HTML 节点内容。如果必须从 `save` 返回原始 HTML ，请使用 `wp.element.RawHTML` 。顾名思义，这容易导致 跨站脚本攻击，因此应尽可能使用 WordPress 元素层次结构。

注意： save 函数应该是一个纯函数且无状态，仅依赖于调用时使用的属性。它不应使用任何 API，如 `useState` 或 `useEffect` ，也不应从其他来源检索信息；例如，不能在内部使用数据模块 - `select( store ).selector( ... )` 。这是因为如果外部信息发生变化，在后续编辑文章时，区块可能会被标记为无效（ 了解更多关于验证的信息）。

如果需要将其他信息作为保存的一部分，开发者可以考虑以下两种替代方案之一：

- 使用 动态区块 并在服务器上动态检索所需信息。
- 将外部值存储为属性，并在区块的 `edit` 函数中随着变化动态更新。

对于 动态区块, `save` 的返回值可以表示区块内容的缓存副本，仅在实现区块的插件被禁用时显示。

如果未指定，默认实现将不会在文章内容中保存动态区块的标记，而是始终在区块显示在网站前端时进行计算
。

区块包装器属性

与 `edit` 函数类似，在渲染静态区块时，务必将 `useBlockProps.save()` 返回的区块属性添加到区块的包装元素上。这不仅能确保区块类名正确渲染，还能包含区块支持 API 注入的任何 HTML 属性。

属性

与 `edit` 类似， `save` 函数也会接收一个包含属性的对象参数，这些属性可被插入到标记中。

```
save: ( { attributes } ) => {
	const blockProps = useBlockProps.save();

	return <div { ...blockProps }>{ attributes.content }</div>;
};
```

保存区块时，需要按照属性源定义指定的相同格式保存属性。如果未指定属性源，属性将被保存到区块的注释
分隔符中。更多详情请参阅 区块属性文档 。

innerBlocks

传递给 `save` 函数的 props 中还有第二个属性 `innerBlocks` 。该属性通常用于内部操作，需要用到它的场景非常少。

`innerBlocks`
在初始化时是一个包含嵌套块对象表示的数组。在那些极少数可能使用此属性的情况下，它可以帮助你调整块
的渲染方式。例如，你可以根据嵌套块的数量或是否存在特定块类型来以不同方式渲染块。

```
save: ( { attributes, innerBlocks } ) => {
	const { className, ...rest } = useBlockProps.save();

	// innerBlocks在初始化时也可能是对象 - React 元素
	const numberOfInnerBlocks = innerBlocks?.length;
	if ( numberOfInnerBlocks > 1 ) {
		className = className + ( className ? ' ' : '' ) + 'more-than-one';
```

```
};
const blockProps =  { ...rest, className };

return <div { ...blockProps }>{ attributes.content }</div>;
};
```

Examples

Here are a couple examples of using attributes, edit, and save all together.

Saving Attributes to Child Elements

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

edit: ( { attributes, setAttributes } ) => {
const blockProps = useBlockProps();
const updateFieldValue = ( val ) => {
setAttributes( { content: val } );
}
return (
<div { ...blockProps }>
<TextControl
__next40pxDefaultSize
label='My Text Field'
value={ attributes.content }
onChange={ updateFieldValue }
/>
</div>
);
```

```
},

save: ( { attributes } ) => {
	const blockProps = useBlockProps.save();

	return <div { ...blockProps }> { attributes.content } </div>;
},
```

通过序列化保存属性

理想情况下，应保存的属性应包含在标记中。然而，有时这并不实际，因此如果未指定属性源，属性将被序列化并保存到块的注释分隔符中。

此示例可能适用于动态块，例如 最新文章块_____ ，它在服务器端渲染标记。尽管如此，保存函数仍然是必需的，但在这种情况下，它仅返回 null ，因为该块不保存来自编辑器的内容。

```
attributes: {
	postsToShow: {
		type: 'number',
	}
},

edit: ( { attributes, setAttributes } ) => {
	const blockProps = useBlockProps();

	return (
		<div { ...blockProps }>
			<TextControl
				__next40pxDefaultSize
				label='Number Posts to Show'
				value={ attributes.postsToShow }
				onChange={ ( val ) => {
					setAttributes( { postsToShow: parseInt( val ) } );
				}}
			/>
		</div>
	);
},
```

```
save: () => {
return null;
}
```

验证

编辑器加载时，会对文章内容中的所有区块进行验证以确定其准确性，从而防止内容丢失。这与区块的保存实现密切相关，因为如果编辑器无法正确恢复区块，用户可能会无意中删除或修改其内容。在编辑器初始化期间，会使用从文章内容解析出的属性重新生成每个区块的已保存标记。如果新生成的标记与文章内容中已存储的标记不匹配，则该区块将被标记为无效。这是因为我们假设除非用户进行编辑，否则标记应与已保存内容保持一致。

如果检测到区块无效，系统将提示用户选择如何处理无效情况：

无效区块提示

点击 尝试恢复区块 按钮将尽可能尝试恢复操作。

点击区块侧面的 “三点 ”菜单会显示三个选项：

- 解决：打开解决区块对话框，其中包含两个按钮：
  - 转换为 **HTML**：保留已保存文章内容中的原始标记，并将区块从其原始类型转换为 HTML 区块类型，使用户能够直接修改 HTML 标记。
  - 转换为区块：保留已保存文章内容中的原始标记，并将区块从其原始类型转换为已验证的区块类型。
- 转换为 **HTML**：保留已保存文章内容中的原始标记，并将区块从其原始类型转换为 HTML 区块类型，使用户能够直接修改 HTML 标记。
- 转换为经典区块：将已保存文章内容中的原始标记视为正确。由于区块将从其原始类型转换为经典区块类型，因此将无法再使用原始区块类型的可用控件来编辑内容。

验证常见问题

区块如何变为无效？

区块失效最常见的两个原因是：

1. 区块代码存在缺陷，导致内容被意外修改。插件开发者请参阅下文关于如何调试区块失效的问题。
2. 您或外部编辑器修改了区块的 HTML 标记，导致其不再被视为正确格式。

我是插件开发者。如何调试区块被标记为无效的问题？

开始调试前，请务必熟悉上文描述的验证步骤，该步骤记录了检测区块是否无效的流程。如果区块重新生成的
标记与文章内容中保存的不匹配，则该区块无效，这通常是由于区块属性从保存内容中解析不正确导致的。

如果您使用 属性来源 ，请确保从标记中提取的属性完全按预期保存，且类型正确（通常为 'string' 或
'number' ）。

当检测到区块无效时，浏览器开发者工具控制台会记录警告信息。该警告将包含标记差异发生具体位置的详细
信息。请仔细查看预期标记与实际标记之间的差异，以定位问题所在。

我更改了区块的 save 行为，现在旧内容包含无效区块。如何修复此问题？

请参阅 已弃用区块 指南，了解如何在有意的标记变更中兼容旧版内容。

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