全局设置与样式 (theme.json)

目录
设计理念
区块编辑器设置
可按区块单独控制设置
样式统一管理
CSS 自定义属性：预设与自定义
规范
版本
设置
启用 UI 控件
Backward compatibility with add_theme_support
预设
Custom
设置示例
样式
顶层样式
Block styles
引用样式
Element styles
元素伪选择器
变体
区块样式变体的 theme.json 片段
以编程方式注册变体样式
customTemplates
templateParts
模式
使用 theme.json 进行开发
常见问题
CSS 自定义属性的命名模式

- [redacted] -- [redacted]
- How settings under "custom" create new CSS Custom Properties
- Global Stylesheet
- [redacted]
- [redacted] blockGap [redacted]
- [redacted]

WordPress 5.8 [redacted] [redacted] WordPress [redacted] `theme.json` [redacted]

[redacted]

[redacted] API [redacted]

[redacted] [redacted] [redacted]

[redacted]

[redacted] `theme.json` [redacted]

- [redacted]
- [redacted]
- [redacted]

[redacted]

[redacted] `theme.json` [redacted]

[redacted]

- [redacted]
- [redacted] UI [redacted]
- [redacted]

[redacted]

通过使用 `theme.json` 文件以结构化方式设置样式属性，区块编辑器能够 "管理 " 来自不同来源（用户、主题和核心 CSS）的 CSS 。例如，如果主题和用户同时为段落设置了字体大小，系统将仅加载用户设置的样式，而忽略主题的样式。

主要优势包括：

- 减少加载的 CSS 数量
- 避免样式优先级冲突

CSS 自定义属性：预设与自定义

某些样式领域会受益于拥有可在整个网站中变化的共享值。

为满足这一需求，我们已开始在某些地方尝试使用 CSS 自定义属性（即 CSS 变量）：

- 预设：由主题声明的 调色板、字体大小或 渐变会被转换为 CSS 自定义属性，并同时在前端和编辑器中入队。

{% codetabs %} {% Input %}

```
{
  "version": 3,
  "settings": {
    "color": {
      "palette": [
        {
          "name": "Black",
          "slug": "black",
          "color": "#000000"
        },
        {
          "name": "White",
          "slug": "white",
          "color": "#ffffff"
        }
      ]
    }
  }
}
```

{% Output %}

```
body {
--wp--preset--color--black: #000000;
--wp--preset--color--white: #ffffff;
}
```

{% end %}

- 自定义属性 ：还存在一种机制用于创建您自己的 CSS 自定义属性。

{% codetabs %} {% Input %}

```
{
"version": 3,
"settings": {
"custom": {
"line-height": {
"body": 1.7,
"heading": 1.3
}
}
}
}
```

{% Output %}

```
body {
--wp--custom--line-height--body: 1.7;
--wp--custom--line-height--heading: 1.3;
}
```

{% end %}

规范

此规范适用于使用此格式的三种不同来源：核心、主题和用户。主题可以通过创建名为 `theme.json` 的文件来覆盖核心的默认设置。用户通过站点编辑器，也将能够通过正在开发中的用户界面来覆盖主题或核心的偏好设置。

```
{
"version": 3,
```

```
{
"settings": {},
"styles": {},
"customTemplates": {},
"templateParts": {}
}
```

版本

此字段描述 `theme.json` 文件的格式。最新版本是 WordPress 6.6 中引入的 版本 3。

当需要进行破坏性变更时，会引入新版本。这允许主题作者选择何时选择接受这些破坏性变更，并将其 theme.json 文件迁移到新格式。

旧版本的 `theme.json` 具有向后兼容性，将继续适用于较新版本的 WordPress 和 Gutenberg 插件。但新功能将在最新版本上开发。

有关更新到最新版本的详细信息，请遵循 迁移到新版本 中的说明。

设置

> Gutenberg 插件扩展了 WordPress 5.8 中可用的设置，以便它们可以与其他 WordPress 版本一起使用，并且在移植到核心之前会经过一个成熟过程。
>
> 下面的选项卡显示了 WordPress 5.8 支持的设置以及 Gutenberg 插件支持的设置。

设置部分具有以下结构：

{% codetabs %} {% WordPress %}

```
{
"version": 3,
"settings": {
  "border": {
    "radius": false,
    "color": false,
    "style": false,
    "width": false
  },
  "color": {
    "custom": true,
```

```
□□□"customDuotone": true,
□□□"customGradient": true,
□□□"duotone": [],
□□□"gradients": [],
□□□"link": false,
□□□"palette": [],
□□□"text": true,
□□□"background": true,
□□□"defaultGradients": true,
□□□"defaultPalette": true
□□},
□□"custom": {},
□□"layout": {
□□□"contentSize": "800px",
□□□"wideSize": "1000px"
□□},
□□"spacing": {
□□□"margin": false,
□□□"padding": false,
□□□"blockGap": null,
□□□"units": [ "px", "em", "rem", "vh", "vw" ]
□□},
□□"typography": {
□□□"customFontSize": true,
□□□"lineHeight": false,
□□□"dropCap": true,
□□□"fluid": false,
□□□"fontStyle": true,
□□□"fontWeight": true,
□□□"letterSpacing": true,
□□□"textDecoration": true,
□□□"textTransform": true,
□□□"fontSizes": [],
□□□"fontFamilies": []
□□},
□□"blocks": {
□□□"core/paragraph": {
□□□□"color": {},
□□□□"custom": {},
□□□□"layout": {},
```

```
        "spacing": {},
        "typography": {}
      },
      "core/heading": {},
      "etc": {}
    }
  }
}
```

{% Gutenberg %}

```
{
  "version": 3,
  "settings": {
    "appearanceTools": false,
    "border": {
      "color": false,
      "radius": false,
      "style": false,
      "width": false
    },
    "color": {
      "background": true,
      "custom": true,
      "customDuotone": true,
      "customGradient": true,
      "defaultGradients": true,
      "defaultPalette": true,
      "duotone": [],
      "gradients": [],
      "link": false,
      "palette": [],
      "text": true
    },
    "custom": {},
    "dimensions": {
      "aspectRatio": false,
      "height": false,
      "minHeight": false,
      "width": false,
```

```
□□},
□□"layout": {
□□□"contentSize": "800px",
□□□"wideSize": "1000px"
□□},
□□"spacing": {
□□□"blockGap": null,
□□□"margin": false,
□□□"padding": false,
□□□"customSpacingSize": true,
□□□"units": [ "px", "em", "rem", "vh", "vw" ],
□□□"spacingScale": {
□□□□"operator": "*",
□□□□"increment": 1.5,
□□□□"steps": 7,
□□□□"mediumStep": 1.5,
□□□□"unit": "rem"
□□□},
□□□"spacingSizes": []
□□},
□□"typography": {
□□□"customFontSize": true,
□□□"dropCap": true,
□□□"fluid": false,
□□□"fontFamilies": [],
□□□"fontSizes": [],
□□□"fontStyle": true,
□□□"fontWeight": true,
□□□"letterSpacing": true,
□□□"lineHeight": false,
□□□"textAlign": true,
□□□"textColumns": false,
□□□"textDecoration": true,
□□□"textIndent": true,
□□□"textTransform": true
□□},
□□"blocks": {
□□□"core/paragraph": {
□□□□"border": {},
□□□□"color": {},
```

```
□□□□"custom": {},
□□□□"layout": {},
□□□□"spacing": {},
□□□□"typography": {}
□□□},
□□□"core/heading": {},
□□□"etc": {}
□□}
□}
}
```

{% end %}

Each block can configure any of these settings separately, providing a more fine-grained control over what exists via `add_theme_support`. The settings declared at the top-level affect to all blocks, unless a particular block overwrites it. It's a way to provide inheritance and configure all blocks at once.

Note, however, that not all settings are relevant for all blocks. The settings section provides an opt-in/opt-out mechanism for themes, but it's the block's responsibility to add support for the features that are relevant to it. For example, if a block doesn't implement the `dropCap` feature, a theme can't enable it for such a block through `theme.json`.

□□ UI □□

□□□□□□□□□□ `appearanceTools` □□□□□□□□□□□□□□ false □□□□□□□□□□□□□□□□□□□□

- □□□ backgroundImage□ backgroundSize
- □□□ color□ radius□ style□ width
- □□□ link
- □□□ aspectRatio□ height□ minHeight□ width
- □□□ sticky
- □□□ blockGap□ margin□ padding
- □□□ lineHeight

Backward compatibility with add_theme_support

To retain backward compatibility, the existing `add_theme_support` declarations that configure the block editor are retrofit in the proper categories for the top-level section. For example, if a theme uses `add_theme_support('disable-custom-colors')`, it'll be the same as setting `settings.color.custom` to `false`. If the `theme.json` contains any settings, these will take precedence over the values declared via `add_theme_support`. This is the complete list of equivalences:

add_theme_support	theme.json setting
`custom-line-height`	Set `typography.lineHeight` to `true`.
`custom-spacing`	Set `spacing.padding` to `true`.
`custom-units`	Provide the list of units via `spacing.units`.
`disable-custom-colors`	Set `color.custom` to `false`.
`disable-custom-font-sizes`	Set `typography.customFontSize` to `false`.
`disable-custom-gradients`	Set `color.customGradient` to `false`.
`editor-color-palette`	Provide the list of colors via `color.palette`.
`editor-font-sizes`	Provide the list of font size via `typography.fontSizes`.
`editor-gradient-presets`	Provide the list of gradients via `color.gradients`.
`editor-spacing-sizes`	Provide the list of spacing sizes via `spacing.spacingSizes`.
`appearance-tools`	Set `appearanceTools` to `true`.
`border`	Set `border: color, radius, style, width` to `true`.
`link-color`	Set `color.link` to `true`.

□□

□□□□□□□□□□□□□□□□□□□□ UI □□□□□□□□□□□□ `theme.json`
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ CSS □□□□□□□□

□□□□ `theme.json` □□□□□□□□

- `color.duotone` □□□□□□□□□□□□□
- `color.gradients` □□□□□□□□□□□□□□□□□□□□□
- `color.palette` □
  - □□□□□□□ 3 □□□ color□ background-color □ border-color□
  - □□□□□□□□□□□□□□□□
- `spacing.spacingSizes` / `spacing.spacingScale` □□□□□□□□□□□□□□□□□
- `typography.fontSizes` □□□□□□□□□□□□□□□□□□□□□
- `typography.fontFamilies` □□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

- □□□□□□ `--wp--preset--{preset-category}--{preset-slug}` □□□ `--wp--preset--color--black`
- □□ `.has-{preset-slug}-{preset-category}` □□□ `.has-black-color` □

{% codetabs %} {% Input %}

```
{
  "version": 3,
  "settings": {
    "color": {
      "duotone": [
        {
          "colors": [ "#000", "#FFF" ],
          "slug": "black-and-white",
          "name": "Black and White"
        }
      ],
      "gradients": [
        {
          "slug": "blush-bordeaux",
          "gradient": "linear-gradient(135deg,rgb(254,205,165) 0%,rgb(254,45,45) 50%,rgb(107,0,62) 100%)",
          "name": "Blush bordeaux"
        },
        {
          "slug": "blush-light-purple",
          "gradient": "linear-gradient(135deg,rgb(255,206,236) 0%,rgb(152,150,240) 100%)",
          "name": "Blush light purple"
        }
      ],
      "palette": [
        {
          "slug": "strong-magenta",
          "color": "#a156b4",
          "name": "Strong magenta"
        },
        {
          "slug": "very-dark-grey",
          "color": "rgb(131, 12, 8)",
          "name": "Very dark grey"
        }
      ]
    },
    "typography": {
      "fontFamilies": [
        {
```

```
□□□□□"fontFamily": "-apple-system,BlinkMacSystemFont,\"Segoe UI\",Roboto,Oxygen-
Sans,Ubuntu,Cantarell, \"Helvetica Neue\",sans-serif",
□□□□□"slug": "system-font",
□□□□□"name": "System Font"
□□□□},
□□□□{
□□□□□"fontFamily": "Helvetica Neue, Helvetica, Arial, sans-serif",
□□□□□"slug": "helvetica-arial",
□□□□□"name": "Helvetica or Arial"
□□□□}
□□□],
□□□"fontSizes": [
□□□□{
□□□□□"slug": "big",
□□□□□"size": 32,
□□□□□"name": "Big"
□□□□},
□□□□{
□□□□□"slug": "x-large",
□□□□□"size": 46,
□□□□□"name": "Large"
□□□□}
□□□]
□□},
□□"spacing": {
□□□"spacingScale": {
□□□□"operator": "*",
□□□□"increment": 1.5,
□□□□"steps": 7,
□□□□"mediumStep": 1.5,
□□□□"unit": "rem"
□□□},
□□□"spacingSizes": [
□□□□{
□□□□□"slug": "40",
□□□□□"size": "1rem",
□□□□□"name": "Small"
□□□□},
□□□□{
□□□□□"slug": "50",
```

```
□□□□□"size": "1.5rem",
□□□□□"name": "Medium"
□□□□},
□□□□{
□□□□□"slug": "60",
□□□□□"size": "2rem",
□□□□□"name": "Large"
□□□□}
□□□]
□□},
□□"blocks": {
□□□"core/group": {
□□□□"color": {
□□□□□"palette": [
□□□□□□□{
□□□□□□□□"slug": "black",
□□□□□□□□"color": "#000000",
□□□□□□□□"name": "Black"
□□□□□□□},
□□□□□□□{
□□□□□□□□"slug": "white",
□□□□□□□□"color": "#ffffff",
□□□□□□□□"name": "White"
□□□□□□□}
□□□□□]
□□□□}
□□□}
□□}
□}
}
```


{% Output %}


```
/* Top-level custom properties */
body {
□--wp--preset--color--strong-magenta: #a156b4;
□--wp--preset--color--very-dark-grey: #444;
□--wp--preset--gradient--blush-bordeaux: linear-gradient( 135deg, rgb( 254, 205, 165 ) 0%, rgb(
254, 45, 45 ) 50%, rgb( 107, 0, 62 ) 100% );
□--wp--preset--gradient--blush-light-purple: linear-gradient( 135deg, rgb( 255, 206, 236 ) 0%,
```

```
rgb( 152, 150, 240 ) 100% );
--wp--preset--font-size--x-large: 46;
--wp--preset--font-size--big: 32;
--wp--preset--font-family--helvetica-arial: Helvetica Neue, Helvetica, Arial, sans-serif;
--wp--preset--font-family--system: -apple-system,BlinkMacSystemFont,\"Segoe
UI\",Roboto,Oxygen-Sans,Ubuntu,Cantarell, \"Helvetica Neue\",sans-serif;
--wp--preset--spacing--20: 0.44rem;
--wp--preset--spacing--30: 0.67rem;
--wp--preset--spacing--40: 1rem;
--wp--preset--spacing--50: 1.5rem;
--wp--preset--spacing--60: 2.25rem;
--wp--preset--spacing--70: 3.38rem;
--wp--preset--spacing--80: 5.06rem;
}

/* Block-level custom properties (bounded to the group block) */
.wp-block-group {
--wp--preset--color--black: #000000;
--wp--preset--color--white: #ffffff;
}

/* Top-level classes */
.has-strong-magenta-color { color: #a156b4 !important; }
.has-strong-magenta-background-color { background-color: #a156b4 !important; }
.has-strong-magenta-border-color { border-color: #a156b4 !important; }
.has-very-dark-grey-color { color: #444 !important; }
.has-very-dark-grey-background-color { background-color: #444 !important; }
.has-very-dark-grey-border-color { border-color: #444 !important; }
.has-blush-bordeaux-background { background: linear-gradient( 135deg, rgb( 254, 205, 165 ) 0%,
rgb( 254, 45, 45 ) 50%, rgb( 107, 0, 62 ) 100% ) !important; }
.has-blush-light-purple-background { background: linear-gradient( 135deg, rgb( 255, 206, 236 )
0%, rgb( 152, 150, 240 ) 100% ) !important; }
.has-big-font-size { font-size: 32; }
.has-normal-font-size { font-size: 16; }

/* Block-level classes (bounded to the group block) */
.wp-block-group.has-black-color { color: #a156b4 !important; }
.wp-block-group.has-black-background-color { background-color: #a156b4 !important; }
.wp-block-group.has-black-border-color { border-color: #a156b4 !important; }
.wp-block-group.has-white-color { color: #444 !important; }
```

```
.wp-block-group.has-white-background-color { background-color: #444 !important; }
.wp-block-group.has-white-border-color { border-color: #444 !important; }
```

{% end %}

To maintain backward compatibility, the presets declared via `add_theme_support` will also generate the CSS Custom Properties. If the `theme.json` contains any presets, these will take precedence over the ones declared via `add_theme_support`.

Preset classes are attached to the content of a post by some user action. That's why the engine will add `!important` to these, because user styles should take precedence over theme styles.

Custom

In addition to create CSS Custom Properties for the presets, the `theme.json` also allows for themes to create their own, so they don't have to be enqueued separately. Any values declared within the `custom` field will be transformed to CSS Custom Properties following this naming schema: `--wp--custom--<variable-name>`.

For example:

{% codetabs %} {% Input %}

```
{
□"version": 3,
□"settings": {
□□"custom": {
□□□"baseFont": 16,
□□□"lineHeight": {
□□□□"small": 1.2,
□□□□"medium": 1.4,
□□□□"large": 1.8
□□□}
□□},
□□"blocks": {
□□□"core/group": {
□□□□"custom": {
□□□□□"baseFont": 32
□□□□}
□□□}
□□}
□}
```

```
}
```

{% Output %}

```
body {
--wp--custom--base-font: 16;
--wp--custom--line-height--small: 1.2;
--wp--custom--line-height--medium: 1.4;
--wp--custom--line-height--large: 1.8;
}
.wp-block-group {
--wp--custom--base-font: 32;
}
```

{% end %}

Note that the name of the variable is created by adding `--` in between each nesting level and `camelCase` fields are transformed to `kebab-case`.

□□□

- □□□□□□□□□□□□□

```
{
"version": 3,
"settings": {
"color": {
"custom": false
},
"blocks": {
"core/paragraph": {
"color": {
"custom": true
}
}
}
}
}
```

- □□□□□□□□□□□

```
{
  "version": 3,
  "settings": {
    "blocks": {
      "core/button": {
        "border": {
          "radius": false
        }
      }
    }
  }
}
```

- □□□□□□□□□□□□□□□□□□□□

```
{
  "version": 3,
  "settings": {
    "color": {
      "palette": [
        {
          "slug": "black",
          "color": "#000000",
          "name": "Black"
        },
        {
          "slug": "white",
          "color": "#FFFFFF",
          "name": "White"
        },
        {
          "slug": "red",
          "color": "#FF0000",
          "name": "Red"
        },
        {
          "slug": "green",
          "color": "#00FF00",
          "name": "Green"
        },
```

```
□□□□{
□□□□□"slug": "blue",
□□□□□"color": "#0000FF",
□□□□□"name": "Blue"
□□□□}
□□□]
□□},
□□"blocks": {
□□□"core/group": {
□□□□"color": {
□□□□□"palette": [
□□□□□□□{
□□□□□□□□"slug": "black",
□□□□□□□□"color": "#000000",
□□□□□□□□"name": "Black"
□□□□□□□},
□□□□□□□{
□□□□□□□□"slug": "white",
□□□□□□□□"color": "#FFF",
□□□□□□□□"name": "White"
□□□□□□□}
□□□□□]
□□□□}
□□□}
□□}
□}
}
```

样式

> Gutenberg 插件扩展了 WordPress 5.8 中可用的样式，因此它们可以与其他 WordPress 版本一起使用，并且在移植到核心之前会经历一个成熟过程。
>
> 下面的选项卡显示了 WordPress 5.8 支持的样式以及 Gutenberg 插件支持的样式。

每个区块都通过 区块支持机制 声明其公开哪些样式属性。这些支持声明用于在编辑器中自动生成该区块的 UI 控件。主题可以通过 `theme.json` 为任何区块使用任何样式属性 —— 主题有责任根据区块标记等验证其是否正常工作。

{% codetabs %} {% WordPress %}

```
{
  "version": 3,
  "styles": {
    "border": {
      "radius": "value",
      "color": "value",
      "style": "value",
      "width": "value"
    },
    "filter": {
      "duotone": "value"
    },
    "color": {
      "background": "value",
      "gradient": "value",
      "text": "value"
    },
    "spacing": {
      "blockGap": "value",
      "margin": {
        "top": "value",
        "right": "value",
        "bottom": "value",
        "left": "value",
      },
      "padding": {
        "top": "value",
        "right": "value",
        "bottom": "value",
        "left": "value"
      }
    },
    "typography": {
      "fontSize": "value",
      "fontStyle": "value",
      "fontWeight": "value",
      "letterSpacing": "value",
      "lineHeight": "value",
      "textDecoration": "value",
      "textTransform": "value"
```

```
□□},
□□"elements": {
□□□"link": {
□□□□"border": {},
□□□□"color": {},
□□□□"spacing": {},
□□□□"typography": {}
□□□},
□□□"h1": {},
□□□"h2": {},
□□□"h3": {},
□□□"h4": {},
□□□"h5": {},
□□□"h6": {}
□□},
□□"blocks": {
□□□"core/group": {
□□□□"border": {},
□□□□"color": {},
□□□□"spacing": {},
□□□□"typography": {},
□□□□"elements": {
□□□□□"link": {},
□□□□□"h1": {},
□□□□□"h2": {},
□□□□□"h3": {},
□□□□□"h4": {},
□□□□□"h5": {},
□□□□□"h6": {}
□□□□}
□□□},
□□□"etc": {}
□□}
□}
}
```

{% Gutenberg %}

```
{
□"version": 3,
```

```
"styles": {
  "border": {
    "color": "value",
    "radius": "value",
    "style": "value",
    "width": "value"
  },
  "color": {
    "background": "value",
    "gradient": "value",
    "text": "value"
  },
  "dimensions": {
    "aspectRatio": "value",
    "height": "value"
    "minHeight": "value"
    "width": "value"
  },
  "filter": {
    "duotone": "value"
  },
  "spacing": {
    "blockGap": "value",
    "margin": {
      "top": "value",
      "right": "value",
      "bottom": "value",
      "left": "value"
    },
    "padding": {
      "top": "value",
      "right": "value",
      "bottom": "value",
      "left": "value"
    }
  },
  "typography": {
    "fontFamily": "value",
    "fontSize": "value",
    "fontStyle": "value",
```

```
□□□"fontWeight": "value",
□□□"letterSpacing": "value",
□□□"lineHeight": "value",
□□□"textColumns": "value",
□□□"textDecoration": "value",
□□□"textTransform": "value"
□□},
□□"elements": {
□□□"link": {
□□□□"border": {},
□□□□"color": {},
□□□□"spacing": {},
□□□□"typography": {}
□□□},
□□□"h1": {},
□□□"h2": {},
□□□"h3": {},
□□□"h4": {},
□□□"h5": {},
□□□"h6": {},
□□□"heading": {},
□□□"button": {},
□□□"caption": {}
□□},
□□"blocks": {
□□□"core/group": {
□□□□"border": {},
□□□□"color": {},
□□□□"dimensions": {},
□□□□"spacing": {},
□□□□"typography": {},
□□□□"elements": {
□□□□□"link": {},
□□□□□"h1": {},
□□□□□"h2": {},
□□□□□"h3": {},
□□□□□"h4": {},
□□□□□"h5": {},
□□□□□"h6": {}
□□□□}
```

```
□□□},
□□□"etc": {}
□□}
□}
}
```

{% end %}

□□□□

□□□□□□□□□□ `body` □□□□□□□□

{% codetabs %} {% Input %}

```
{
□"version": 3,
□"styles": {
□□"color": {
□□□"text": "var(--wp--preset--color--primary)"
□□}
□}
}
```

{% Output %}

```
body {
□color: var( --wp--preset--color--primary );
}
```

{% end %}

Block styles

Styles found within a block will be enqueued using the block selector.

By default, the block selector is generated based on its name such as `.wp-block-<blockname-without-namespace>`. For example, `.wp-block-group` for the `core/group` block. There are some blocks that want to opt-out from this default behavior. They can do so by explicitly telling the system which selector to use for them via the `__experimentalSelector` key within the `supports` section of its `block.json` file. Note that the block needs to be registered server-side for the `__experimentalSelector` field to be available to the style engine.

{% codetabs %} {% Input %}

```
{
□"version": 3,
□"styles": {
□□"color": {
□□□"text": "var(--wp--preset--color--primary)"
□□},
□□"blocks": {
□□□"core/paragraph": {
□□□□"color": {
□□□□□"text": "var(--wp--preset--color--secondary)"
□□□□}
□□□},
□□□"core/group": {
□□□□"color": {
□□□□□"text": "var(--wp--preset--color--tertiary)"
□□□□}
□□□}
□□}
□}
}
```

{% Output %}

```
body {
□color: var( --wp--preset--color--primary );
}
p { /* The core/paragraph opts out from the default behaviour and uses p as a selector. */
□color: var( --wp--preset--color--secondary );
}
.wp-block-group {
□color: var( --wp--preset--color--tertiary );
}
```

{% end %}

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Gutenberg □□□ □□□□□

`styles.color.background` □□□□□□□□□□□□

```
"styles": {
□□"color": {
□□□"background": "var(--wp--preset--color--primary)"
□□}
□}
```

□□□□□ `ref: "styles.color.background"` □□□□□□□□□□

```
{
□"color": {
□□"text": { "ref": "styles.color.background" }
□}
}
```

Element styles

In addition to top-level and block-level styles, there's the concept of elements that can be used in both places. There's a closed set of them:

Supported by Gutenberg:

- `button`: maps to the `wp-element-button` CSS class. Also maps to `wp-block-button__link` for backwards compatibility.
- `caption`: maps to the `.wp-element-caption, .wp-block-audio figcaption, .wp-block-embed figcaption, .wp-block-gallery figcaption, .wp-block-image figcaption, .wp-block-table figcaption, .wp-block-video figcaption` CSS classes.
- `heading`: maps to all headings, the `h1 to h6` CSS selectors.

Supported by WordPress:

- `h1`: maps to the `h1` CSS selector.
- `h2`: maps to the `h2` CSS selector.
- `h3`: maps to the `h3` CSS selector.
- `h4`: maps to the `h4` CSS selector.
- `h5`: maps to the `h5` CSS selector.
- `h6`: maps to the `h6` CSS selector.
- `link`: maps to the `a` CSS selector.

If they're found in the top-level the element selector will be used. If they're found within a block, the selector to be used will be the element's appended to the corresponding block.

{% codetabs %} {% Input %}

```
{
  "version": 3,
  "styles": {
    "typography": {
      "fontSize": "var(--wp--preset--font-size--normal)"
    },
    "elements": {
      "h1": {
        "typography": {
          "fontSize": "var(--wp--preset--font-size--huge)"
        }
      },
      "h2": {
        "typography": {
          "fontSize": "var(--wp--preset--font-size--big)"
        }
      },
      "h3": {
        "typography": {
          "fontSize": "var(--wp--preset--font-size--medium)"
        }
      }
    },
    "blocks": {
      "core/group": {
        "elements": {
          "h2": {
            "typography": {
              "fontSize": "var(--wp--preset--font-size--small)"
            }
          },
          "h3": {
            "typography": {
              "fontSize": "var(--wp--preset--font-size--smaller)"
            }
          }
        }
      }
    }
  }
}
```

```
}
```

{% Output %}

```
body {
□font-size: var( --wp--preset--font-size--normal );
}
h1 {
□font-size: var( --wp--preset--font-size--huge );
}
h2 {
□font-size: var( --wp--preset--font-size--big );
}
h3 {
□font-size: var( --wp--preset--font-size--medium );
}
.wp-block-group h2 {
□font-size: var( --wp--preset--font-size--small );
}
.wp-block-group h3 {
□font-size: var( --wp--preset--font-size--smaller );
}
```

{% end %}

□□□□□□

Gutenberg □□□□□□ `:hover`□ `:focus`□ `:focus-visible`□ `:visited`□ `:active`□ `:link`□ `:any-link`□

```
"elements": {
□□"link": {
□□□"color": {
□□□□"text": "green"
□□□},
□□□":hover": {
□□□□"color": {
□□□□□"text": "hotpink"
□□□□}
□□□}
□□}
```

```
}
```

变体

区块可以拥有 "样式变体"，其定义遵循 block.json 规范。主题作者可以使用 `theme.json` 文件为现有样式变体定义样式属性。未注册样式变体的样式将被忽略。

请注意，变体是一个 "区块概念"——它们仅在绑定到区块时才存在。`theme.json` 规范尊重这一区别，只允许在区块层级定义 `variations`，而非顶层。同样值得强调的是，只有定义在区块的 `block.json` 文件中或通过服务器端的 `register_block_style` 注册的变体，才会被 `theme.json` 视为 "已注册" 的样式变体。

例如，以下是如何为 `core/quote` 区块现有的 `plain` 变体提供样式：

```
{
"version": 3,
"styles": {
"blocks": {
"core/quote": {
"variations": {
"plain": {
"color": {
"background": "red"
}
}
}
}
}
}
}
```

生成的 CSS 输出为：

```
.wp-block-quote.is-style-plain {
background-color: red;
}
```

多个区块类型也可以共享相同的变体样式。有两种推荐的方法来定义此类共享样式：

1. `theme.json` 部分文件
2. 通过编程方式，使用 `register_block_style`

区块样式变体的 theme.json 片段

与主题样式变体片段类似，区块样式变体的片段也位于主题的 `/styles` 目录中。然而，它们通过引入一个名为 `blockTypes` 的顶级属性来与主题样式变体区分开来。`blockTypes` 属性是一个数组，包含了已注册该区块样式变体的区块类型。

此外，还提供了一个 `slug` 属性，以确保定义区块样式变体的不同来源之间的一致性，并将 `slug` 与可翻译的 `title` 属性解耦。

以下是一个 `theme.json` 片段的示例，它为 Group、Columns 和 Media & Text 区块类型定义了 "Variation A" 区块样式的样式：

```
{
  "$schema": "https://schemas.wp.org/trunk/theme.json",
  "version": 3,
  "title": "Variation A",
  "slug": "variation-a",
  "blockTypes": [ "core/group", "core/columns", "core/media-text" ],
  "styles": {
    "color": {
      "background": "#eed8d3",
      "text": "#201819"
    },
    "elements": {
      "heading": {
        "color": {
          "text": "#201819"
        }
      }
    },
    "blocks": {
      "core/group": {
        "color": {
          "background": "#825f58",
          "text": "#eed8d3"
        },
        "elements": {
          "heading": {
            "color": {
              "text": "#eed8d3"
            }
```

```
□□□□□}
□□□□}
□□□}
□□}
□}
}
```

以编程方式注册变体样式

除了使用 `theme.json` 片段外，你还可以在通过 `register_block_style` 注册变体本身的同时注册变体样式。这是通过为一系列区块类型注册区块样式，同时在 `style_data` 选项中传递一个 "样式对象 "来实现的。

下面的示例为 "群组 "和 "栏目 "区块注册了一个 "绿色 "变体。请注意，通过 `style_data` 传递的样式对象遵循与 `theme.json` 片段中 `styles` 属性相同的结构。

```php
register_block_style(
    array( 'core/group', 'core/columns' ),
    array(
        'name'       => 'green',
        'label'      => __( 'Green' ),
        'style_data' => array(
            'color'    => array(
                'background' => '#4f6f52',
                'text'       => '#d2e3c8',
            ),
            'blocks'   => array(
                'core/group' => array(
                    'color' => array(
                        'background' => '#739072',
                        'text'       => '#e3eedd',
                    ),
                ),
            ),
            'elements' => array(
                'link'   => array(
                    'color'  => array(
                        'text' => '#ead196',
                    ),
                    ':hover' => array(
                        'color' => array(
```

```
                        'text' => '#ebd9b4',
                    ),
                ),
            ),
        ),
    )
);
```

customTemplates

> WordPress 5.9

`templates` `my-custom-template.html` `theme.json`

- name
- title
- postTypes `page`

```
{
    "version": 3,
"customTemplates": [
  {
    "name": "my-custom-template",
    "title": "The template title",
    "postTypes": [
      "page",
      "post",
      "my-cpt"
    ]
  }
]
}
```

templateParts

> WordPress 5.9

在此字段中，主题可以列出 `parts` 文件夹中存在的模板部件。例如，对于一个名为 `my-template-part.html` 的模板部件，`theme.json` 可以声明该模板部件实体的区域术语，该实体负责在编辑器中渲染相应的区块变体（页眉区块、页脚区块等）。在 json 中定义此区域术语将使该设置在该模板部件实体的所有使用中保持一致，这与仅影响单个区块的区块属性不同。不建议将区域定义为区块属性，因为这仅在“幕后”用于弥合占位符流程与实体创建之间的差距。

目前，区块变体仅针对区域术语的 "header" 和 "footer" 值存在，任何其他值以及未在 json 中定义的模板部件将默认为通用模板部件区块。变体将在编辑器界面中通过特定图标表示，包装器将默认使用相应的语义 HTML 元素（这也可以通过模板部件区块上设置的 `tagName` 属性覆盖），并将使模板部件情境化，以便在未来的编辑器改进中实现更自定义的流程。

- name: 必需。
- title: 可选，可翻译。
- area: 可选，默认设置为 `uncategorized` 且不会触发区块变体。

```
{
    "version": 3,
	"templateParts": [
		{
			"name": "my-template-part",
			"title": "Header",
			"area": "header"
		}
	]
}
```

模式

> 自 WordPress 6.0 版本起支持。

在此字段中，主题可以列出要从 模式目录 注册的模式。 `patterns` 字段是一个包含模式目录中模式 `slugs` 的数组。模式 slug 可以通过模式目录中单个模式视图的 `url` 提取。例如，在 URL `https://wordpress.org/patterns/pattern/partner-logos` 中，slug 是 `partner-logos`。

```
{
	"version": 3,
	"patterns": [ "short-text-surrounded-by-round-images", "partner-logos" ]
}
```

使用 theme.json 进行开发
在开发过程中，很难记住 theme.json 的设置、属性以及不同 WordPress 版本支持哪些设置，因此使用提供的
JSON 模式会很有帮助。

许多代码编辑器都支持 JSON 模式，并能在编辑器中直接提供工具提示、自动完成或模式验证等帮助。

每个 WordPress 版本的 theme.json 模式都可在 https://schemas.wp.org/wp/{{version}}/theme.json 获取。例如， WordPress 5.8 的模式位于 https://schemas.wp.org/wp/5.8/theme.json
。为确保仅使用对用户可用的功能，最好使用主题支持的最旧版本。

包含 Gutenberg 插件所有最新更改的最新模式可在 https://schemas.wp.org/trunk/theme.json 获取。

请查阅编辑器的文档以了解 JSON 模式支持情况。例如，在 Visual Studio Code 中，您需要在 theme.json 文件的顶层属性中添加 "$schema": "https://schemas.wp.org/wp/x.x/theme.json"
，但其他编辑器的配置方式可能不同。

使用模式进行验证的示例

常见问题

CSS 自定义属性的命名模式

您可能已经注意到系统创建的 CSS 自定义属性所使用的命名模式，包括使用双连字符 -- 来分隔不同的 "概念 " 。请看以下示例。

预设 如 --wp--preset--color--black 可以分解为以下几个部分：

- - wp ：命名空间 CSS 变量的前缀。
preset：表示这是一个属于预设的 CSS 变量。
color：表示该变量属于哪个预设类别。可以是 color、 font-size、 gradients。
black：特定预设值的 slug。

自定义 属性如 --wp--custom--line-height--body，可以分解为以下几个部分：

- - wp ：命名空间 CSS 变量的前缀。
custom ：表示这是由主题创建的 "自由格式 " CSS 变量。
line-height--body：将 "custom"对象键转换为字符串的结果。

- - 作为分隔符有两个作用：

- 可读性，便于人类理解。可以将其视为类似于 BEM 命名模式，它分隔了 "类别 "。
- 可解析性，便于机器理解。使用定义好的结构允许机器理解属性 `--wp--preset--color--black` 的含义：它是一个绑定到 slug 为 "black" 的颜色预设的值，这为我们提供了更多处理它们的空间。

为何使用 `--` 作为分隔符？

我们本可以选择其他分隔符，例如单个 `-`。

但这会带来问题，因为除非强制主题作者不在变量名中使用 `-`，否则无法确定如何将 `--wp-custom-line-height-template-header` 转换回对象。

通过保留 `--` 作为类别分隔符，并允许主题作者使用 `-` 作为单词边界，命名更加清晰：`--wp--custom--line-height--template-header`。

How settings under "custom" create new CSS Custom Properties

The algorithm to create CSS Variables out of the settings under the "custom" key works this way:

This is for clarity, but also because we want a mechanism to parse back a variable name such `--wp--custom--line-height--body` to its object form in theme.json. We use the same separation for presets.

For example:

{% codetabs %} {% Input %}

```
{
□"version": 3,
□"settings": {
□□"custom": {
□□□"lineHeight": {
□□□□"body": 1.7
□□□},
□□□"font-primary": "-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen-Sans, Ubuntu,
Cantarell, 'Helvetica Neue', sans-serif"
□□}
□}
}
```

{% Output %}

```
body {
--wp--custom--line-height--body: 1.7;
--wp--custom--font-primary: "-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen-Sans, Ubuntu, Cantarell, 'Helvetica Neue', sans-serif";
}
```

{% end %}

A few notes about this process:

- `camelCased` keys are transformed into its `kebab-case` form, as to follow the CSS property naming schema. Example: `lineHeight` is transformed into `line-height`.
- Keys at different depth levels are separated by `--`. That's why `line-height` and `body` are separated by `--`.
- You shouldn't use `--` in the names of the keys within the `custom` object. Example, **don't do** this:

```
{
"version": 3,
"settings": {
"custom": {
"line--height": { // DO NOT DO THIS
"body": 1.7
}
}
}
}
```

Global Stylesheet

In WordPress 5.8, the CSS for some of the presets defined by WordPress (font sizes, colors, and gradients) was loaded twice for most themes: in the block-library stylesheet plus in the global stylesheet. Additionally, there were slight differences in the CSS in both places.

In WordPress 5.9 release, CSS of presets are consolidated into the global stylesheet, that is now loaded for all themes. Each preset value generates a single CSS Custom Property and a class, as in:

```
/* CSS Custom Properties for the preset values */
body {
  --wp--preset--<PRESET_TYPE>--<PRESET_SLUG>: <DEFAULT_VALUE>;
  --wp--preset--color--pale-pink: #f78da7;
```

```
  --wp--preset--font-size--large: 36px;
  /* etc. */
}

/* CSS classes for the preset values */
.has-<PRESET_SLUG>-<PRESET_TYPE> { ... }
.has-pale-pink-color { color: var(--wp--preset--color--pale-pink) !important; }
.has-large-font-size { font-size: var(--wp--preset--font-size--large) !important; }
```

For themes to override the default values they can use the `theme.json` and provide the same slug. Themes that do not use a `theme.json` can still override the default values by enqueuing some CSS that sets the corresponding CSS Custom Property.

`Example` (sets a new value for the default large font size):

```
body {
 --wp--preset--font-size--large: <NEW_VALUE>;
}
```

用户提供的链接颜色特异性

在 WordPress 5.8 中，当用户为特定区块选择链接颜色时，我们以 `.wp-element-<ID>` 的形式为该区块附加一个类，然后排队加载以下样式：

```
.wp-element-<ID> a { color: <USER_COLOR_VALUE> !important; }
```

虽然这始终保留了用户偏好，但特异性过强，并与某些区块中合法使用不应被视为链接的 HTML 元素发生冲突。为了 解决此问题，在 WordPress 5.9 版本中，移除了 `!important` 并更新了相应区块，以使用高于用户链接颜色的特异性来设置 a 元素的样式，现在样式为：

```
.wp-element-<ID> a { color: <USER_COLOR_VALUE>; }
```

由于此更改，现在区块作者和主题作者有责任确保始终尊重用户选择，并且用户提供的链接颜色（特异性 011）不会被覆盖。

什么是 blockGap 以及如何使用它?

对于包含内部区块的区块（如 Group、Columns、Buttons 和 Social Icons），`blockGap` 控制内部区块之间的间距。要使 `blockGap` 生效，该区块还必须启用 `layout` 区块支持，该支持提供了可通过区块间距控件调整的布局样式。根据区块的布局，`blockGap` 值将作为垂直边距或 `gap`

值输出。在编辑器中，`blockGap` 值的控件称为 区块间距 ，位于尺寸面板中。

```
{
□"version": 3,
□"settings": {
□□"spacing": {
□□□"blockGap": true,
□□}
□},
□"styles": {
□□"spacing": {
□□□"blockGap": "1.5rem"
□□}
□}
}
```

`blockGap` 的设置可以是布尔值或 `null` 值，默认为 `null`。这允许对样式输出进行额外级别的控制。
`theme.json` 文件中的 `settings.spacing.blockGap` 设置接受以下值：

- `true`：选择在编辑器 UI 中显示 区块间距 控件并输出 `blockGap` 样式。
- `false`：选择不在编辑器 UI 中显示 区块间距 控件，但存储在 `theme.json` 中的 `blockGap` 样式仍会被渲染。这允许主题使用 `blockGap` 值，同时不允许用户在编辑器内进行更改。
- `null`（默认）：选择不在编辑器 UI 中显示 区块间距 控件， 并且 阻止输出 `blockGap` 样式。

为根 `styles.spacing.blockGap` 样式定义的值也会作为一个 CSS 属性输出，名为 `--wp--style--block-gap`。

为什么浏览器中样式更新需要这么长时间?

当您使用 theme.json 进行开发时，可能会注意到更改需要 30 秒以上才能在浏览器中显示，这是因为 `theme.json` 被缓存了。要解决此缓存问题，请在 `wp-config.php` 中将 `WP_DEBUG` 或 `SCRIPT_DEBUG` 设置为 'true'。这会告诉 WordPress 跳过缓存并始终使用最新数据。

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