When To Use

Used when the user needs to make a customized color selection.

Examples

Basic Usage

```tsx
import React from 'react';
import { ColorPicker } from 'antd';

const Demo = () => <ColorPicker defaultValue="#1677ff" />;

export default Demo;
```

Trigger size

```tsx
import React from 'react';
import { ColorPicker, Space } from 'antd';

const Demo = () => (
  <Space>
    <Space direction="vertical">
      <ColorPicker defaultValue="#1677ff" size="small" />
      <ColorPicker defaultValue="#1677ff" />
      <ColorPicker defaultValue="#1677ff" size="large" />
    </Space>
    <Space direction="vertical">
      <ColorPicker defaultValue="#1677ff" size="small" showText />
      <ColorPicker defaultValue="#1677ff" showText />
      <ColorPicker defaultValue="#1677ff" size="large" showText />
    </Space>
  </Space>
);

export default Demo;
```

controlled mode

```tsx
import React, { useState } from 'react';
import { ColorPicker, Space } from 'antd';
import type { ColorPickerProps, GetProp } from 'antd';

type Color = GetProp<ColorPickerProps, 'value'>;
```

```tsx
const Demo: React.FC = () => {
  const [color, setColor] = useState<Color>('#1677ff');

  return (
    <Space>
      <ColorPicker value={color} onChange={setColor} />
      <ColorPicker value={color} onChangeComplete={setColor} />
    </Space>
  );
};

export default Demo;
```

Line Gradient

v5.20.0

```tsx
import React from 'react';
import { ColorPicker, Space } from 'antd';

const DEFAULT_COLOR = [
  {
    color: 'rgb(16, 142, 233)',
    percent: 0,
  },
  {
    color: 'rgb(135, 208, 104)',
    percent: 100,
  },
];

const Demo = () => (
  <Space direction="vertical">
    <ColorPicker
      defaultValue={DEFAULT_COLOR}
      allowClear
      showText
      mode={['single', 'gradient']}
      onChangeComplete={(color) => {
        console.log(color.toCssString());
      }}
    />
    <ColorPicker
      defaultValue={DEFAULT_COLOR}
      allowClear
      showText
```

```
      mode="gradient"
      onChangeComplete={(color) => {
        console.log(color.toCssString());
      }}
    />
  </Space>
);

export default Demo;
```

Rendering Trigger Text

```
import React, { useState } from 'react';
import { DownOutlined } from '@ant-design/icons';
import { ColorPicker, Space } from 'antd';

const Demo = () => {
  const [open, setOpen] = useState(false);
  return (
    <Space direction="vertical">
      <ColorPicker defaultValue="#1677ff" showText allowClear />
      <ColorPicker
        defaultValue="#1677ff"
        showText={(color) => <span>Custom Text ({color.toHexString()})
</span>}
      />
      <ColorPicker
        defaultValue="#1677ff"
        open={open}
        onOpenChange={setOpen}
        showText={() => (
          <DownOutlined
            rotate={open ? 180 : 0}
            style={{
              color: 'rgba(0, 0, 0, 0.25)',
            }}
          />
        )}
      />
    </Space>
  );
};

export default Demo;
```

Disable

```tsx
import React from 'react';
import { ColorPicker } from 'antd';

export default () => <ColorPicker defaultValue="#1677ff" showText disabled
/>;
```

Disabled Alpha

```tsx
import React from 'react';
import { ColorPicker } from 'antd';

const Demo = () => <ColorPicker defaultValue="#1677ff" disabledAlpha />;

export default Demo;
```

Clear Color

```tsx
import React from 'react';
import { ColorPicker } from 'antd';

export default () => {
  const [color, setColor] = React.useState<string>('#1677ff');

  return (
    <ColorPicker
      value={color}
      allowClear
      onChange={(c) => {
        setColor(c.toHexString());
      }}
    />
  );
};
```

Custom Trigger

```tsx
import React, { useMemo, useState } from 'react';
import { Button, ColorPicker } from 'antd';
import type { ColorPickerProps, GetProp } from 'antd';

type Color = Extract<GetProp<ColorPickerProps, 'value'>, string | {
```

```
cleared: any }>;

const Demo: React.FC = () => {
  const [color, setColor] = useState<Color>('#1677ff');

  const bgColor = useMemo<string>(
    () => (typeof color === 'string' ? color : color!.toHexString()),
    [color],
  );

  const btnStyle: React.CSSProperties = {
    backgroundColor: bgColor,
  };

  return (
    <ColorPicker value={color} onChange={setColor}>
      <Button type="primary" style={btnStyle}>
        open
      </Button>
    </ColorPicker>
  );
};

export default Demo;
```

Custom Trigger Event

```
import React from 'react';
import { ColorPicker } from 'antd';

const Demo = () => <ColorPicker defaultValue="#1677ff" trigger="hover" />;

export default Demo;
```

Color Format

```
import React, { useState } from 'react';
import { ColorPicker, Space } from 'antd';
import type { ColorPickerProps, GetProp } from 'antd';

type Color = Extract<GetProp<ColorPickerProps, 'value'>, string | {
cleared: any }>;
type Format = GetProp<ColorPickerProps, 'format'>;

const HexCase: React.FC = () => {
```

```
  const [colorHex, setColorHex] = useState<Color>('#1677ff');
  const [formatHex, setFormatHex] = useState<Format | undefined>('hex');

  const hexString = React.useMemo<string>(
    () => (typeof colorHex === 'string' ? colorHex :
colorHex?.toHexString()),
    [colorHex],
  );

  return (
    <Space>
      <ColorPicker
        format={formatHex}
        value={colorHex}
        onChange={setColorHex}
        onFormatChange={setFormatHex}
      />
      <span>HEX: {hexString}</span>
    </Space>
  );
};

const HsbCase: React.FC = () => {
  const [colorHsb, setColorHsb] = useState<Color>('hsb(215, 91%, 100%)');
  const [formatHsb, setFormatHsb] = useState<ColorPickerProps['format']>
('hsb');

  const hsbString = React.useMemo(
    () => (typeof colorHsb === 'string' ? colorHsb :
colorHsb?.toHsbString()),
    [colorHsb],
  );

  return (
    <Space>
      <ColorPicker
        format={formatHsb}
        value={colorHsb}
        onChange={setColorHsb}
        onFormatChange={setFormatHsb}
      />
      <span>HSB: {hsbString}</span>
    </Space>
  );
};
```

```
const RgbCase: React.FC = () => {
  const [colorRgb, setColorRgb] = useState<Color>('rgb(22, 119, 255)');
  const [formatRgb, setFormatRgb] = useState<ColorPickerProps['format']>
('rgb');

  const rgbString = React.useMemo(
    () => (typeof colorRgb === 'string' ? colorRgb :
colorRgb?.toRgbString()),
    [colorRgb],
  );

  return (
    <Space>
      <ColorPicker
        format={formatRgb}
        value={colorRgb}
        onChange={setColorRgb}
        onFormatChange={setFormatRgb}
      />
      <span>RGB: {rgbString}</span>
    </Space>
  );
};

const Demo: React.FC = () => (
  <Space direction="vertical" size="middle" style={{ display: 'flex' }}>
    <HexCase />
    <HsbCase />
    <RgbCase />
  </Space>
);

export default Demo;
```

Preset Colors

```
import React from 'react';
import { generate, green, presetPalettes, red } from '@ant-design/colors';
import { ColorPicker, theme } from 'antd';
import type { ColorPickerProps } from 'antd';

type Presets = Required<ColorPickerProps>['presets'][number];

function genPresets(presets = presetPalettes) {
  return Object.entries(presets).map<Presets>(([label, colors]) => ({
label, colors, key: label }));
```

```
}

const Demo: React.FC = () => {
  const { token } = theme.useToken();
  const presets = genPresets({ primary: generate(token.colorPrimary), red,
green });
  return <ColorPicker presets={presets} defaultValue="#1677ff" />;
};

export default Demo;
```

Custom Render Panel

```
import React from 'react';
import { cyan, generate, green, presetPalettes, red } from '@ant-
design/colors';
import { Col, ColorPicker, Divider, Row, Space, theme } from 'antd';
import type { ColorPickerProps } from 'antd';

type Presets = Required<ColorPickerProps>['presets'][number];

function genPresets(presets = presetPalettes) {
  return Object.entries(presets).map<Presets>(([label, colors]) => ({
label, colors, key: label }));
}

const HorizontalLayoutDemo = () => {
  const { token } = theme.useToken();

  const presets = genPresets({
    primary: generate(token.colorPrimary),
    red,
    green,
    cyan,
  });

  const customPanelRender: ColorPickerProps['panelRender'] = (
    _,
    { components: { Picker, Presets } },
  ) => (
    <Row justify="space-between" wrap={false}>
      <Col span={12}>
        <Presets />
      </Col>
      <Divider type="vertical" style={{ height: 'auto' }} />
      <Col flex="auto">
```

```
        <Picker />
      </Col>
    </Row>
  );

  return (
    <ColorPicker
      defaultValue={token.colorPrimary}
      styles={{ popupOverlayInner: { width: 480 } }}
      presets={presets}
      panelRender={customPanelRender}
    />
  );
};

const BasicDemo = () => (
  <ColorPicker
    defaultValue="#1677ff"
    panelRender={(panel) => (
      <div className="custom-panel">
        <div
          style={{
            fontSize: 12,
            color: 'rgba(0, 0, 0, 0.88)',
            lineHeight: '20px',
            marginBottom: 8,
          }}
        >
          Color Picker
        </div>
        {panel}
      </div>
    )}
  />
);

export default () => (
  <Space direction="vertical">
    <Space>
      <span>Add title:</span>
      <BasicDemo />
    </Space>
    <Space>
      <span>Horizontal layout:</span>
      <HorizontalLayoutDemo />
    </Space>
```

```
  </Space>
);
```

Pure Render

Debug

```tsx
import React, { useState } from 'react';
import { ColorPicker } from 'antd';
import type { ColorPickerProps, GetProp } from 'antd';

const { _InternalPanelDoNotUseOrYouWillBeFired: PureRenderColorPicker } =
ColorPicker;

type Color = GetProp<ColorPickerProps, 'value'>;

const Demo: React.FC = () => {
  const [color, setColor] = useState<Color>('#1677ff');
  return (
    <div style={{ paddingInlineStart: 100 }}>
      <PureRenderColorPicker value={color} onChange={setColor} />
    </div>
  );
};

export default Demo;
```

API

Common props ref：Common props

> *This component is available since* `antd@5.5.0`.

Property	Description	Type	Default	Versio
allowClear	Allow clearing color selected	boolean	false	
arrow	Configuration for popup arrow	`boolean | { pointAtCenter: boolean }`	true	
children	Trigger of ColorPicker	React.ReactNode	-	
defaultValue	Default value of color	string | `Color`	-	

defaultFormat	Default format of color	`rgb` | `hex` | `hsb`	hex	5.9.0
disabled	Disable ColorPicker	boolean	-	
disabledAlpha	Disable Alpha	boolean	-	5.8.0
disabledFormat	Disable format of color	boolean	-	
destroyTooltipOnHide	Whether destroy popover when hidden	`boolean`	false	5.7.0
format	Format of color	`rgb` | `hex` | `hsb`	-	
mode	Configure single or gradient color	`'single' | 'gradient' | ('single' | 'gradient')[]`	`single`	5.20.0
open	Whether to show popup	boolean	-	
presets	Preset colors	`{ label: ReactNode, colors: Array<string | Color>, defaultOpen?: boolean, key?: React.Key }[]`	-	defau 5.11. 5.23.
placement	Placement of popup	The design of the placement parameter is the same as the `Tooltips` component.	`bottomLeft`	
panelRender	Custom Render Panel	`(panel: React.ReactNode, extra: { components: { Picker: FC; Presets: FC } }) => React.ReactNode`	-	5.7.0

showText	Show color text	boolean | (color: Color) => React.ReactNode	-	5.7.0
size	Setting the trigger size	large | middle | small	middle	5.7.0
trigger	ColorPicker trigger mode	hover | click	click	
value	Value of color	string | Color	-	
onChange	Callback when value is changed	(value: Color, css: string) => void	-	
onChangeComplete	Called when color pick ends. Will not change the display color when value controlled by onChangeComplete	(value: Color) => void	-	5.7.0
onFormatChange	Callback when format is changed	(format: 'hex' | 'rgb' | 'hsb') => void	-	
onOpenChange	Callback when open is changed	(open: boolean) => void	-	
onClear	Called when clear	() => void	-	5.6.0

Color

Property	Description	Type	Version
toCssString	Convert to CSS support format	() => string	5.20.0
toHex	Convert to hex format characters, the return type like: 1677ff	() => string	-
toHexString	Convert to hex format color string, the return type like: #1677ff	() => string	-
toHsb	Convert to hsb object	() => ({ h: number, s: number, b: number, a number })	-
toHsbString	Convert to hsb format color string, the return type like: hsb(215,	() => string	-

	91%, 100%)		
toRgb	Convert to rgb object	() => ({ r: number, g: number, b: number, a number })	-
toRgbString	Convert to rgb format color string, the return type like: rgb(22, 119, 255)	() => string	-

FAQ

Questions about color assignment

The value of the color selector supports both string color values and selector-generated `Color` objects. However, since there is a precision error when converting color strings of different formats to each other, it is recommended to use selector-generated `Color` objects for assignment operations in controlled scenarios, so that the precision problem can be avoided and the values are guaranteed to be accurate and the selector can work as expected.