[深入 JSX 2](#_Toc22190)

[指定 React 元素类型 2](#_Toc23660)

[指定属性值 2](#_Toc5681)

[子代 3](#_Toc31443)

[传递子代 3](#_Toc32333)

[使用 PropTypes 进行类型检查 3](#_Toc27182)

[限制单个子代 4](#_Toc25675)

[属性默认值 4](#_Toc6823)

[静态类型检查 4](#_Toc12199)

[Flow 4](#_Toc5595)

[在项目中添加 Flow 4](#_Toc19289)

[从编译过的代码中剥离 Flow 语法 5](#_Toc11139)

[运行 Flow 5](#_Toc7754)

[添加 Flow 类型注释 5](#_Toc3220)

[TypeScript 5](#_Toc11399)

[在一个项目中添加 TypeScript 5](#_Toc4140)

[配置 TypeScript 编译器 6](#_Toc20207)

[运行 TypeScript 6](#_Toc30977)

# 深入 JSX

JSX为 React.createElement(component, props, ...children)方法提供的语法糖。

## 指定 React 元素类型

JSX 的标签名决定了 React 元素的类型。

大写开头的 JSX 标签表示自定义或导入的 React 组件。这些标签将会被编译为同名变量并被引用，所以如果你使用了 <Foo /> 表达式，则必须在作用域中先声明 Foo 变量。

大写开头的 JSX 标签表示内置组件，如 <div>、<span>。

JSX 编译后会调用 React.createElement 方法，所以JSX 代码中必须首先声明 React 变量。

可使用 JSX 中的点表示法来引用 React 组件，如：MyComponents.DatePicker。

在运行时选择类型，不能使用表达式来作为 React 元素的标签。如果需通过表达式来确定 React 元素的类型，需要先将类型赋值给大写开头的变量，再以大写变量创建标签。

## 指定属性值

#### JavaScript 表达式

使用{} 包裹JavaScript 表达式作为一个属性值；

<MyComponent foo={1 + 2 + 3 + 4} /> //foo=10

#### **字符串常量**

可将字符串常量作为属性值传递。该属性值会被解析为HTML非转义字符串。

<MyComponent message="hello world" />

<MyComponent message={'hello world'} />

#### **默认为 True**

如果你没有给属性传值，它默认为 true。

#### **扩展属性**

可使用 ... 作为扩展操作符，来解构传递整个属性对象

const props = {firstName: 'Ben', lastName: 'Hector'};

<Greeting {...props} />;

## 子代

在包含开始和结束标签的 JSX 表达式中，标记之间的内容作为特殊的参数传递：props.children。

### 传递子代

#### **字符串常量**

在开始和结束标签之间放入一个字符串，则 props.children 就是那个字符串。

JSX 会移除空行和开始与结尾处的空格；字符串常量内部的换行会被压缩成一个空格。

#### **JSX**

可通过子代嵌入更多的 JSX 元素；React 组件可通过数组的形式返回多个元素。

#### **JavaScript** 表达式

可将任何 {} 包裹的 JavaScript 表达式作为子代传递。

#### **函数**

使用自定义组件，调用 props.children 来获得传递的子代。

#### **布尔**值、Null 和 Undefined 被忽略

false、null、undefined 和 true 都是有效的子代，但它们不会直接被渲染。

根据条件来确定是否渲染React元素：

{showHeader && <Header />}

如需显示false、true、null、undefined，必须转换成字符串。

String(myVariable)

# 使用 PropTypes 进行类型检查

注意: React.PropTypes 自 React v15.5 起已弃用。请使用 prop-types 库代替。

propTypes 只在开发模式下进行检查。

Greeting.propTypes = {

name: PropTypes.string //根据不同需求使用不同的验证器

};

## 限制单个子代

Greeting.propTypes = {

children: PropTypes.element.isRequired

};

## 属性默认值

通过配置 defaultProps 为 props定义默认值。

Greeting.defaultProps = {

name: 'Stranger'

};

# 静态类型检查

静态类型检查器可以在运行代码之前识别某些类型的问题。对于更大的代码库我们建议使用Flow 或者TypeScript等静态类型检查器。

## Flow

一个针对 JavaScript 代码的静态类型检查器，通过使用特殊的语法为类型注释扩展了 JavaScript语言。

### 在项目中添加 Flow

Yarn方式：

yarn add --dev flow-bin

yarn run flow init

npm方式：

npm install --save-dev flow-bin //安装最新版的 Flow

npm run flow init //创建Flow 配置文件

将 flow 添加到 package.json中的 "scripts" 部分。

{

// ...

"scripts": {

"flow": "flow",

// ...

},

// ...

}

### 从编译过的代码中剥离 Flow 语法

Create React App，Flow被默认剥离。

#### Babel剥离Flow

安装一个特殊的 preset

yarn add --dev babel-preset-flow

npm install --save-dev babel-preset-flow

.babelrc 文件，将 flow preset 加入Babel 配置。

{

"presets": [

"flow",

"react"

]

}

### 运行 Flow

yarn flow

npm run flow

### 添加 Flow 类型注释

默认情况下， Flow 仅检查包含"// @flow"的文件。

[Flow 文档：类型注释](https://flow.org/en/docs/types/)

[Flow 文档：编辑器](https://flow.org/en/docs/editors/)

[Flow 文档： React](https://flow.org/en/docs/react/)

[Linting in Flow](https://medium.com/flow-type/linting-in-flow-7709d7a7e969)

## TypeScript

### 在一个项目中添加 TypeScript

yarn add --dev typescript

npm install --save-dev typescript

配置tsc到 package.json 中的 “scripts” 部分。

"scripts": {

"build": "tsc",

// ...

}

### 配置 TypeScript 编译器

生成tsconfig.json文件，并配置选项。

tsc --init

文件扩展名

.ts，默认的文件扩展名，

.tsx，为包含 JSX 代码使用的特殊扩展名。

### 运行 TypeScript

yarn build

npm run build

TypeScript 的更多信息：

[TypeScript 文档：基本类型](https://www.typescriptlang.org/docs/handbook/basic-types.html)

[TypeScript 文档：从 Javascript 迁徙](http://www.typescriptlang.org/docs/handbook/migrating-from-javascript.html)

[TypeScript 文档： React 和 Webpack](http://www.typescriptlang.org/docs/handbook/react-&-webpack.html)