要在组件中进行类型检测，你可以使用 propTypes 属性。

使用示例：

import PropTypes from 'prop-types';

class Greeting extends React.Component {

  render() {

    return (

      <h1>Hello, {this.props.name}</h1>

    );

  }

}

Greeting.propTypes = {

// props有一个字段为name，name的类型为string

  name: PropTypes.string

};

我们可以使用PropTypes.string 语句。当给 prop 传递了一个不正确的值时，JavaScript控制台将会显示一条警告。出于性能的原因，propTypes 仅在开发模式中检测。

### 默认的 props 值

你可以通过赋值特定的 defaultProps 属性为 props 定义默认值：

class Greeting extends React.Component {

render() {

return (

<h1>Hello, {this.props.name}</h1>

);

}

}

// 指定 props 的默认值：

Greeting.defaultProps = {

name: 'Stranger'

};

// 渲染为 "Hello, Stranger":

ReactDOM.render(

<Greeting />,

document.getElementById('example')

);

**详细的PropTypes类型**

原文地址：<https://www.cnblogs.com/cangqinglang/p/9077904.html>

MyComponent.propTypes = {

    // 你可以声明一个 prop 是一个特定的 JS 原始类型。

    // 默认情况下，这些都是可选的。

    optionalArray: PropTypes.array,

    optionalBool: PropTypes.bool,

    optionalFunc: PropTypes.func,

    optionalNumber: PropTypes.number,

    optionalObject: PropTypes.object,

    optionalString: PropTypes.string,

    optionalSymbol: PropTypes.symbol,

    // 任何东西都可以被渲染:numbers, strings, elements,或者是包含这些类型的数组(或者是片段)。

optionalNode: PropTypes.node,

    // 一个 React 元素。

optionalElement: PropTypes.element,

    // 你也可以声明一个 prop 是类的一个实例。

    // 使用 JS 的 instanceof 运算符。

optionalMessage: PropTypes.instanceOf(Message),

    // 你可以声明 prop 是特定的值，类似于枚举

optionalEnum: PropTypes.oneOf(['News', 'Photos']),

    // 一个对象可以是多种类型其中之一

    optionalUnion: PropTypes.oneOfType([

        PropTypes.string,

        PropTypes.number,

        PropTypes.instanceOf(Message)

]),

    // 一个某种类型的数组

optionalArrayOf: PropTypes.arrayOf(PropTypes.number),

    // 属性值为某种类型的对象

optionalObjectOf: PropTypes.objectOf(PropTypes.number),

    // 一个特定形式的对象

    optionalObjectWithShape: PropTypes.shape({

        color: PropTypes.string,

        fontSize: PropTypes.number

}),

    // 你可以使用 `isRequired' 链接上述任何一个，以确保在没有提供 prop 的情况下显示警告。

requiredFunc: PropTypes.func.isRequired,

    // 任何数据类型的值

requiredAny: PropTypes.any.isRequired,

    // 你也可以声明自定义的验证器。如果验证失败返回 Error 对象。不要使用 `console.warn` 或者 throw ，

    // 因为这不会在 `oneOfType` 类型的验证器中起作用。

    customProp: function (props, propName, componentName) {

        if (!/matchme/.test(props[propName])) {

            return new Error(

                'Invalid prop `' + propName + '` supplied to' +

                ' `' + componentName + '`. Validation failed.'

            );

        }

},

    // 也可以声明`arrayOf`和`objectOf`类型的验证器，如果验证失败需要返回Error对象。

    // 会在数组或者对象的每一个元素上调用验证器。验证器的前两个参数分别是数组或者对象本身，

    // 以及当前元素的键值。

    customArrayProp: PropTypes.arrayOf(function (propValue, key, componentName, location, propFullName) {

        if (!/matchme/.test(propValue[key])) {

            return new Error(

                'Invalid prop `' + propFullName + '` supplied to' +

                ' `' + componentName + '`. Validation failed.'

            );

        }

    })

};