Typechecking With PropTypes

> **注意:**
> React.PropTypes は React v15.5 において別パッケージに移動しました。代わりに `prop-types` ライブラリを利用するようにしてください。
> コードを自動で変換するための codemod スクリプトを提供しています。

アプリケーションが成長するにつれて、型チェックによって多くの不具合を見つけられるようになります。アプリケーションによっては、Flow もしくは TypeScript のような JavaScript 拡張を使ってアプリケーション全体の型チェックを行うことができるでしょう。しかしそれらを使用せずとも、React は組み込みの型チェック機能を備えています。コンポーネントの props に型チェックを行うために、特別な `propTypes` プロパティを割当てることがます。

```
import PropTypes from 'prop-types';

class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

`PropTypes` は受け取ったデータが有効かどうかを確認するために使用できる種々のバリデーターをエクスポートしています。上記の例では、`PropTypes.string` を使用しています。無効な値がプロパティに与えられた場合、JavaScript のコンソールに警告文が出力されます。パフォーマンス上の理由から、`propTypes` のチェックは開発モードでのみ行われます。

PropTypes

PropTypes によって提供されている様々なバリデーターの実例を紹介します。

```
import PropTypes from 'prop-types';

MyComponent.propTypes = {
  // You can declare that a prop is a specific JS type. By default, these
  // are all optional.
  optionalArray: PropTypes.array,
  optionalBool: PropTypes.bool,
  optionalFunc: PropTypes.func,
  optionalNumber: PropTypes.number,
  optionalObject: PropTypes.object,
  optionalString: PropTypes.string,
  optionalSymbol: PropTypes.symbol,

  // Anything that can be rendered: numbers, strings, elements or an array
  // (or fragment) containing these types.
  optionalNode: PropTypes.node,

  // A React element.
  optionalElement: PropTypes.element,

  // You can also declare that a prop is an instance of a class. This uses
  // JS's instanceof operator.
  optionalMessage: PropTypes.instanceOf(Message),

  // You can ensure that your prop is limited to specific values by treating
  // it as an enum.
  optionalEnum: PropTypes.oneOf(['News', 'Photos']),

  // An object that could be one of many types
  optionalUnion: PropTypes.oneOfType([
    PropTypes.string,
    PropTypes.number,
    PropTypes.instanceOf(Message)
  ]),

  // An array of a certain type
  optionalArrayOf: PropTypes.arrayOf(PropTypes.number),

  // An object with property values of a certain type
  optionalObjectOf: PropTypes.objectOf(PropTypes.number),

  // An object taking on a particular shape
  optionalObjectWithShape: PropTypes.shape({
```

```
    color: PropTypes.string,
    fontSize: PropTypes.number
  }),

  // You can chain any of the above with `isRequired` to make sure a warning
  // is shown if the prop isn't provided.
  requiredFunc: PropTypes.func.isRequired,

  // A value of any data type
  requiredAny: PropTypes.any.isRequired,

  // You can also specify a custom validator. It should return an Error
  // object if the validation fails. Don't `console.warn` or throw, as this
  // won't work inside `oneOfType`.
  customProp: function(props, propName, componentName) {
    if (!/matchme/.test(props[propName])) {
      return new Error(
        'Invalid prop `' + propName + '` supplied to' +
        ' `' + componentName + '`. Validation failed.'
      );
    }
  },

  // You can also supply a custom validator to `arrayOf` and `objectOf`.
  // It should return an Error object if the validation fails. The validator
  // will be called for each key in the array or object. The first two
  // arguments of the validator are the array or object itself, and the
  // current item's key.
  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.'
      );
    }
  })
};
```

単一の子要素を要求する

`PropTypes.element` を使うことで、コンポーネントに単一の子要素しか渡せないことを指定することができます。

```
import PropTypes from 'prop-types';

class MyComponent extends React.Component {
  render() {
    // This must be exactly one element or it will warn.
    const children = this.props.children;
    return (
      <div>
        {children}
      </div>
    );
  }
}

MyComponent.propTypes = {
  children: PropTypes.element.isRequired
};
```

props のデフォルト値

`defaultProps` というプロパティを割り当てることで、`props` に値が渡されなかった際のデフォルト値を定義することができます。

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

// Specifies the default values for props:
Greeting.defaultProps = {
  name: 'Stranger'
};

// Renders "Hello, Stranger":
ReactDOM.render(
  <Greeting />,
  document.getElementById('example')
);
```

もし、transform-class-properties のような Babel 変換のプラグインを使用している場合、`defaultProps` をクラス内の static プロパティとして定義することができます。この記法はまだ TC39 によって確定されていないため、ブラウザ上で正しく表示させるには、Babel 等で変換する必要があります。詳細は class fields proposal を参照してください。

```
class Greeting extends React.Component {
  static defaultProps = {
    name: 'stranger'
  }

  render() {
    return (
      <div>Hello, {this.props.name}</div>
    )
  }
}
```

`defaultProps` を使えば、`this.props.name` が親コンポーネントから値が指定されなかった場合でも値が代入されていることを保証できます。`propTypes` による型チェックは `defaultProps` が解決した後に行われるため、`defaultProps` にも型チェックが適用されます。

このページを編集する