Test Renderer

インポート

```
import TestRenderer from 'react-test-renderer'; // ES6
const TestRenderer = require('react-test-renderer'); // ES5 with npm
```

概要

このパッケージは、DOM やネイティブのモバイル環境に依存せずに React コンポーネントをピュアな JavaScript オブジェクトにレンダーすることができる React レンダラを提供します。
本質的にこのパッケージは、ブラウザや jsdom を利用しなくても、React DOM や React Native コンポーネントがレンダーする（DOM ツリーに似た）ビューの階層構造のスナップショットを容易に取得できるようにするためのものです。
例：

```
import TestRenderer from 'react-test-renderer';

function Link(props) {
  return <a href={props.page}>{props.children}</a>;
}

const testRenderer = TestRenderer.create(
  <Link page="https://www.facebook.com/">Facebook</Link>
);

console.log(testRenderer.toJSON());
// { type: 'a',
//   props: { href: 'https://www.facebook.com/' },
//   children: [ 'Facebook' ] }
```

JSON ツリーを自動的にファイルに保存し、変更が起こったかをテストで確認するには、Jest のスナップショットテスト機能が利用できます。詳細について知る
出力を走査して特定のノードを検索し、それらに対してアサーションを行うこともできます。

```
import TestRenderer from 'react-test-renderer';

function MyComponent() {
  return (
    <div>
      <SubComponent foo="bar" />
      <p className="my">Hello</p>
    </div>
  )
}

function SubComponent() {
  return (
    <p className="sub">Sub</p>
  );
}

const testRenderer = TestRenderer.create(<MyComponent />);
const testInstance = testRenderer.root;

expect(testInstance.findByType(SubComponent).props.foo).toBe('bar');
expect(testInstance.findByProps({className: "sub"}).children).toEqual(['Sub']);
```

TestRenderer

- `TestRenderer.create()`

TestRenderer instance

- `testRenderer.toJSON()`
- `testRenderer.toTree()`
- `testRenderer.update()`
- `testRenderer.unmount()`
- `testRenderer.getInstance()`

- `testRenderer.root`

TestInstance

- `testInstance.find()`
- `testInstance.findByType()`
- `testInstance.findByProps()`
- `testInstance.findAll()`
- `testInstance.findAllByType()`
- `testInstance.findAllByProps()`
- `testInstance.instance`
- `testInstance.type`
- `testInstance.props`
- `testInstance.parent`
- `testInstance.children`

リファレンス

`TestRenderer.create()`

```
TestRenderer.create(element, options);
```

渡された React 要素から `TestRenderer` インスタンスを作成します。実際の DOM は使用しませんが、コンポーネントを完全な形でメモリにレンダーするので、アサーションを行うことができます。返されたインスタンスは、次のメソッドとプロパティを持ちます。

`testRenderer.toJSON()`

```
testRenderer.toJSON()
```

レンダーされたツリーを表すオブジェクトを返します。このツリーは `<div>` もしくは `<View>` のようなプラットフォーム固有のノードとそのプロパティを含みますが、ユーザー定義のコンポーネントは含まれません。スナップショットテストに便利です。

`testRenderer.toTree()`

```
testRenderer.toTree()
```

レンダーされたツリーを表すオブジェクトを返します。`toJSON()` とは異なり、このツリーはより詳細なものであり、ユーザー定義のコンポーネントも含んでいます。テストレンダラを利用して自作のアサーションライブラリを作成している場合以外は、恐らくこのメソッドが必要となることはないでしょう。

`testRenderer.update()`

```
testRenderer.update(element)
```

メモリ上のツリーを新規のルート要素で再レンダーします。ルートでの React の更新をシミュレートします。新しい要素が以前の要素と同じ型と key を持つ場合は、ツリーは更新されます。それ以外の場合は新しいツリーを再マウントします。

`testRenderer.unmount()`

```
testRenderer.unmount()
```

メモリ上のツリーをアンマウントし、適切なライフサイクルイベントを発生させます。

`testRenderer.getInstance()`

```
testRenderer.getInstance()
```

ルート要素に対応したインスタンスがある場合はそれを返します。関数コンポーネントはインスタンスを持たないため、ルート要素が関数コンポーネントの場合、このメソッドはうまく動作しません。

testRenderer.root

```
testRenderer.root
```

ツリー上の特定のノードに対してアサーションを行う際に役立つ、ルート「テストインスタンス」を返します。これは、配下の他の「テストインスタンス」を検索する際に使用することができます。

testInstance.find()

```
testInstance.find(test)
```

`test(testInstance)` が `true` を返す単一の子テストインスタンスを検索します。もし `test(testInstance)` に対して `true` を返すテストインスタンスの数がちょうど 1 でない場合は、エラーがスローされます。

testInstance.findByType()

```
testInstance.findByType(type)
```

与えられた `type` を持つ単一の子テストインスタンスを検索します。もし与えられた `type` を持つテストインスタンスの数がちょうど 1 でない場合、エラーがスローされます。

testInstance.findByProps()

```
testInstance.findByProps(props)
```

与えられた `props` を持つ単一の子テストインスタンスを検索します。もし与えられた `props` を持つテストインスタンスの数がちょうど 1 でない場合、エラーがスローされます。

testInstance.findAll()

```
testInstance.findAll(test)
```

`test(testInstance)` が `true` を返す全ての子テストインスタンスを検索します。

testInstance.findAllByType()

```
testInstance.findAllByType(type)
```

与えられた `type` を持つ全ての子テストインスタンスを検索します。

testInstance.findAllByProps()

```
testInstance.findAllByProps(props)
```

与えられた `props` を持つ全ての子テストインスタンスを検索します。

testInstance.instance

```
testInstance.instance
```

当該テストインスタンスに対応するコンポーネントのインスタンスです。関数コンポーネントはインスタンスを持たないため、クラスコンポーネントでのみ使用することができます。与えられたコンポーネント内での `this` の値と一致します。

testInstance.type

```
testInstance.type
```

当該テストインスタンスに対応するコンポーネントの型です。例えば、`<Button />` コンポーネントは `Button` 型を持っています。

`testInstance.props`

```
testInstance.props
```

当該テストインスタンスに対応するコンポーネントの props です。例えば、`<Button size="small" />` コンポーネントは `{size: 'small'}` を props として持っています。

`testInstance.parent`

```
testInstance.parent
```

当該テストインスタンスの親テストインスタンスです。

`testInstance.children`

```
testInstance.children
```

当該テストインスタンスの子テストインスタンスです。

使い方の一例

オプションとして `createNodeMock` 関数を `TestRenderer.create` に渡すことで、独自のモック refs を作成することができます。`createNodeMock` は現在の要素を受け取り、モックの ref オブジェクトを返す必要があります。refs に依存したコンポーネントのテストに便利です。

```
import TestRenderer from 'react-test-renderer';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.input = null;
  }
  componentDidMount() {
    this.input.focus();
  }
  render() {
    return <input type="text" ref={el => this.input = el} />
  }
}

let focused = false;
TestRenderer.create(
  <MyComponent />,
  {
    createNodeMock: (element) => {
      if (element.type === 'input') {
        // mock a focus function
        return {
          focus: () => {
            focused = true;
          }
        };
      }
      return null;
    }
  }
);
expect(focused).toBe(true);
```

このページを編集する