コードベースの概要

このセクションでは React コードベースの構成や規則そして実装についての概要を説明します。
あなたが React にコントリビュートしたい場合に、このガイドがあなたがより快適に変更を行えるように手助けとなる事を願っています。
これらの規約のいずれかをあなたの React アプリケーションで推奨しているというわけでは必ずしもありません。規約の多くは歴史的な理由で存在しており、時間とともに変化する可能性があります。

外部の依存関係

React はほとんど外部の依存関係を持っていません。通常、 `require()` は React 自身のコードベースのファイルを指します。ただし、比較的にまれな例外がいくつかあります。
fbjs リポジトリは React がいくつかの小さなユーティリティを Relay のようなライブラリと共有し、それらと同期を保つために存在しています。Facebook のエンジニアが必要に応じて変更を行えるようであってほしいので、Node のエコシステムにある同等の小さなモジュール群には依存しません。fbjs 内にあるユーティリティは全て非公開 API であり、それらは React のような Facebook プロジェクトでのみ使用されるためのものです。

最上位フォルダ

React リポジトリをクローンした後、プロジェクトのルートディレクトリに複数のフォルダがあることに気がつくでしょう：

- `packages` には React リポジトリの全てのパッケージのメタデータ（`package.json` など）や、ソースコード（`src` サブディレクトリ）が含まれています。**あなたの変更がコードに関するものなら、各パッケージの `src` サブディレクトリが作業時間のほとんどを過ごす場所となります。**
- `fixtures` にはコントリビューター向けの React の小さなテスト用アプリケーションが含まれています。
- `build` は React のビルド出力です。リポジトリには存在しませんが、最初に React をビルドした後に clone した React ディレクトリに現れます。

ドキュメントは React 本体とは異なるリポジトリにホスティングされています。
他にも最上位にあるフォルダがありますが、ほとんどはツールが使用するためのもので、コントリビュートする場合にそれらに直面することはほぼありません。

テストコードをソースコードのそばに格納する

ユニットテスト用の最上位ディレクトリはありません。代わりに、テスト対象のファイルのすぐ隣の `__tests__` というディレクトリに置いています。
例えば、`setInnerHTML.js` のテストは、すぐ隣の `__tests__/setInnerHTML-test.js` にあります。

警告と不変数 (Invariant)

React のコードベースは警告の表示に `warning` モジュールを使用します：

```
var warning = require('warning');

warning(
  2 + 2 === 4,
  'Math is not working today.'
);
```

警告は `warning` 内の条件が `false` の時に表示されます。

この警告の出力条件についての 1 つの考え方は、例外に対応する条件を書くのではなく、正常の状況を反映する条件を書くべきだ、ということです。
重複した警告文をコンソールにスパムすることを避けることをお勧めします：

```
var warning = require('warning');

var didWarnAboutMath = false;
if (!didWarnAboutMath) {
  warning(
    2 + 2 === 4,
    'Math is not working today.'
  );
  didWarnAboutMath = true;
}
```

警告は開発時にのみ有効になります。本番時には警告は完全に取り除かれます。コードの一部の実行を禁止する必要がある場合は、代わりに `invariant` モジュールを使用してください：

```
var invariant = require('invariant');

invariant(
  2 + 2 === 4,
  'You shall not pass!'
);
```

invariant は `invariant` 内の条件が `false` の時スローされます。

“Invariant” とは “この条件は常に true である” ということの単なる言い換えです。アサーションを作成していると考えることができます。

開発バージョンと本番バージョンで同じ動作となるようにしておく事は重要ですので、 `invariant` は両方の環境でスローされます。本番バージョンではエラーメッセージは自動的にエラーコードに変換され、バイト数に悪影響が出ることを避けます。

開発バージョンと本番バージョン

コードベース内で `__DEV__` 擬似グローバル変数を使用して、開発用のコードブロックを保護することができます。
コンパイル時にインライン化され、CommonJS のビルド時には `process.env.NODE_ENV !== 'production'` というチェックに変換されます。
スタンドアロンなビルドにおいては、minify されないビルドでは `true` となり、minify されたビルドではそれを囲む `if` ブロックごと完全に除去されます。

```
if (__DEV__) {
  // This code will only run in development.
}
```

Flow

最近私たちはコードベースに Flow によるチェックを導入し始めました。ライセンスヘッダで `@flow` アノテーションがマークされているファイルは型チェックをされています。
既存のコードに Flow アノテーションを追加するプルリクエストを受けつけています。Flow アノテーションは以下のようになります：

```
ReactRef.detachRefs = function(
  instance: ReactInstance,
  element: ReactElement | string | number | null | false,
): void {
  // ...
}
```

可能であれば、新しいコードは Flow アノテーションを使うべきです。 `yarn flow` を実行することでローカル環境で Flow を使ってコードをチェックできます。

動的な依存性注入

React は動的な依存性注入を一部のモジュールで使用しています。注入は常に明示的ですが、残念ながらこれはコードの理解の妨げにもなっています。この注入は React が元々 DOM のみをターゲットとしてサポートしていたために存在しています。React Native は React のフォークプロジェクトとして始まりました。React Native に動作の一部をオーバーライドさせるために動的な依存性注入が必要だったのです。
以下のように動的に依存性を宣言するモジュールを見かけるかもしれません：

```
// Dynamically injected
var textComponentClass = null;

// Relies on dynamically injected value
function createInstanceForText(text) {
  return new textComponentClass(text);
}

var ReactHostComponent = {
  createInstanceForText,

  // Provides an opportunity for dynamic injection
  injection: {
    injectTextComponentClass: function(componentClass) {
      textComponentClass = componentClass;
    },
  },
};

module.exports = ReactHostComponent;
```

`injection` フィールドは何らかの特殊な方法で処理されることはありません。しかし慣例として、実行時にこのモジュールに（おそらくプラットフォーム固有の）依存性を注入する必要があることを表します。
コードベースには複数の依存性注入ポイントがあります。将来的には、動的注入メカニズムを取り除き、ビルド中にすべての部品を静的に結合する予定です。

複数のパッケージ

React は単一リポジトリ (monorepo) です。そのリポジトリには複数の別々のパッケージが含まれているので、それらの変更はまとめて調整でき、issue も 1 箇所にまとまっています。

React コア (core)

React の「コア (core)」は最上位の React API を全て含んでいます。例えば：

- `React.createElement()`
- `React.Component`
- `React.Children`

React core にはコンポーネントの定義に必要な API のみが含まれています。リコンシリエーションのアルゴリズムやあらゆるプラットフォーム固有のコードは含まれません。コアは React DOM や React Native コンポーネントに使われています。
React core のコードはソースツリーの `packages/react` に格納されています。npm では `react` パッケージとして利用可能です。対応するスタンドアロンなブラウザ向けのビルドは `react.js` と呼ばれ、`React` というグローバル変数をエクスポートします。

レンダラ

React は元々 DOM のために作成されましたが、後になって React Native によりネイティブなプラットフォームもサポートするようになりました。これにより React の内部に “レンダラ (renderer)” の概念が導入されました。

レンダラは React ツリーを、基盤となるプラットフォーム固有の呼び出しへと変換する方法を管理します。

レンダラは `packages/` にも格納されています：

- React DOM Renderer は React コンポーネントを DOM に変換します。トップレベルの `ReactDOM` API を実装し、npm では `react-dom` パッケージとして利用可能です。`react-dom.js` と呼ばれるスタンドアロンなブラウザ向けバンドルとしても使用でき、`ReactDOM` というグローバル変数をエクスポートします。
- React Native Renderer は React コンポーネントをネイティブのビューに変換します。React Native の内部で使われています。
- React Test Renderer は React コンポーネントを JSON ツリーに変換します。Jest のスナップショットテスト機能で使用され、npm では react-test-renderer パッケージとして利用可能です。

その他に公式にサポートしているレンダラは `react-art` だけです。個別の GitHub リポジトリにありましたが、現在はメインのソースツリーに移動しました。

> **補足：**
> 技術的に、`react-native-renderer` は React に React Native の実装とやり取りの方法を教える非常に薄い層です。ネイティブのビューを管理する実際のプラットフォーム固有のコードは、コンポーネントと共に React Native リポジトリ にあります。

リコンサイラ (reconciler)

React DOM と React Native のように大幅に異なるレンダラでも多くのロジックを共有する必要があります。特に、ツリーのリコンシリエーションのアルゴリズムは可能なかぎり同じものにして、宣言型レンダリング、独自コンポーネント、state、ライフサイクルメソッドおよび ref がプラットフォーム間で一貫した動作となるようにするべきです。

異なるレンダラ間でコードの一部を共有することでこの課題を解決します。React のこの箇所を “リコンサイラ（reconciler, 差分検出処理）” と呼んでいます。`setState()` のような更新がスケジュールされると、差分検出処理ではツリー内のコンポーネントで `render()` を呼び出して、コンポーネントをマウント、更新、もしくはアンマウントします。

リコンサイラは、パブリック API がないため、個別にパッケージ化されていません。代わりに、それらは React DOM や React Native などのレンダラーによってのみ使用されます。

スタック (stack) リコンサイラ

“stack” リコンサイラは React 15 およびそれ以前で実装されています。それ以降は使用されていませんが、次のセクションでは詳細に説明されています。

Fiber リコンサイラ

“fiber” リコンサイラは stack リコンサイラが内在的に抱える問題を解決し、いくつかの長年の問題を修正することを目的とした新しい取り組みです。React 16 以降はデフォルトのリコンサイラとなっています。

主な目的は以下の通りです：

- 中断可能な作業を小分けに分割する機能
- 進行中の作業に優先順位を付けたり、再配置や再利用をする機能
- React でレイアウトをサポートするための親子間を行き来しながらレンダーする機能
- render() から複数の要素を返す機能
- error boundary のサポートの向上

React Fiber のアーキテクチャに関してここやここで読むことができます。React 16 と共にリリースされていますが、非同期機能についてはデフォルトではまだ有効化されていません。

ソースコードは `packages/react-reconciler` に格納されています。

イベントシステム

React は React DOM と React Native の両方で動作し、レンダラに依存しない合成 (synthetic) イベントシステムを実装しています。ソースコードは `packages/events` に格納されています。

このイベントシステムのコードについて深く掘り下げた動画（66 分）があります。

この次は？

React 16 以前のリコンサイラについてより詳細に学ぶには次のセクションを読んでください。新しいリコンサイラの内部についてはまだドキュメント化していません。

このページを編集する