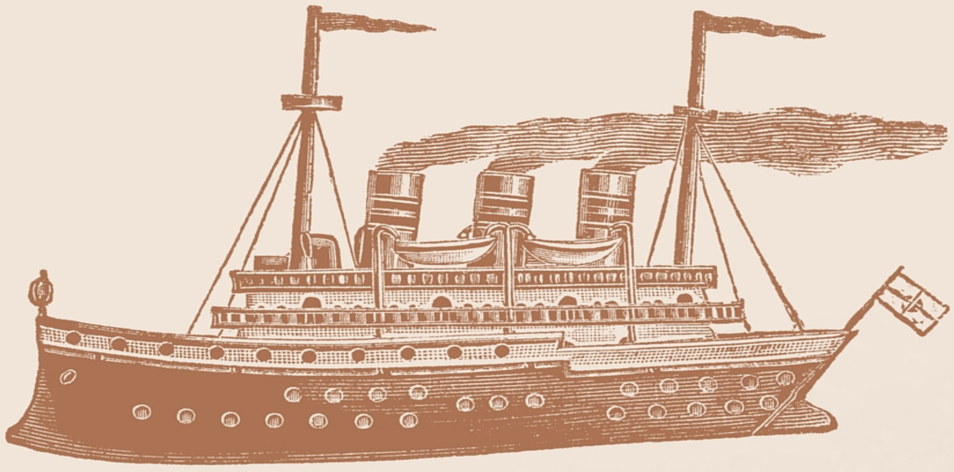


The React Native BOOK

TRUONG HOANG DUNG

The React Native Book

Truong Hoang Dung

This book is for sale at http://leanpub.com/thereactnativebook

This version was published on 2015-06-08

Tweet This Book!

Please help Truong Hoang Dung by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought The React Native Book. A very highly recommended book for everyone.

The suggested hashtag for this book is #reactnative.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#reactnative

Also By Truong Hoang Dung

Contents

Introduction

This book is a practical and quick guide to React-Native framework.

You don't need to be an Objective-C master to learn. So after reading this book, i'm sure you can bring your idea to Apple Store in 1 day.

What you will learn:

- Learn how to setup and debug a React Native application
- Learn how to architect an iOS application
- Learn how to layout the iOS interface
- Learn how to use basic React Native/iOS controls and modules
- Learn how to use Node.js to build your own API server
- Learn how to drawing native Canvas on iOS device
- Learn how to perform HTTP requests with polyfills
- Learn how to play music with custom modules
- Learn how to use SQLite as your database
- Learn how to test the components
- Learn Flux, Store and some of best practices using React components

How to read this book ?

If you come from Web development, this book is perfect for your need. All the content brings the Web development concepts to iOS development through React Native technology.

But don't worry, instead of pointing you to reference sites to learn a concept, the book teaches you all from scratch. At beginning, i thought that i just need to write a `recipe-like` ebook that just guide you the steps needed to write some specific applications and maybe it's enough for you. But in the process of writing, my mind has changed also. The philosophy of React/React-Native is `learn once, write everywhere`, so it's nessessary for you to really learn and understand all those concepts before you master all the technology behind React-Native and develop your own iOS apps.

The good news for you is, after next 5 months, the Android version for ReactNative will be released, so you can naturally get all the benefits from this book to develop Android application also.

About the structure of all chapters:

- Each chapter begins with concepts from Web Development. And you will learn it by doing on Web Development.
- Then there is a part on transition to React Native development. And you will know which part will be brought to, which part is fixed, and which part is not needed anymore.
- The final of the book will use all the knowledges you've learnt to develop real applications.

What will be updated in this book ?

- All the updates from ReactNative development.
- All new libraries to use through ReactNative community.
- The book itself: When there is something that need to be changed, it will be changed.
- The Android chapters (In future after the releases in next months)

You will get all these updates for free after you buy this book once.

Installation and setup new project

If you go to main React-native site, you can see these steps below to setup: ## Requirements

```
OS X - This repo only contains the iOS implementation right now, and Xcode only \
runs on Mac.
New to Xcode? Download it from the Mac App Store.
Homebrew is the recommended way to install node, watchman, and flow.
brew install node.
brew install --HEAD watchman. We recommend installing watchman, otherwise you mi\
ght hit a node file watching bug.
brew install flow. If you want to use flow.
```

Quick start

```
npm install -g react-native-cli
react-native init AwesomeProject
cd AwesomeProject
```

In the newly created folder AwesomeProject/

```
Open AwesomeProject.xcodeproj and hit run in Xcode
Open index.ios.js in your text editor of choice and edit some lines
Hit cmd+R (twice) in your iOS simulator to reload the app and see your change!
```

Congratulations! You've just successfully run and modified your first React Native app.

A closer look at application creation

Let's look what's inside the AwesomeProject/ just created:

```
AwesomeProject.xcodeproj/
iOS/
index.ios.js
node_modules/
package.json
```

To give you the full picture, here is what exactly happens when running `react-native init AwesomeProject`:

- A folder named AwesomeProject is created
- In it, the package.json file is created
- npm install –save react-native is run, which installs react-native and its dependencies into AwesomeProject/node_modules and declares react-native as a dependency of your project in AwesomeProject/package.json
- The globally installed react-native CLI tool then hands over control to the local CLI tool that has just been installed into AwesomeProject/node_modules/react-native/local-cli/cli.js
- This in turn executes AwesomeProject/node_modules/react-native/init.sh, which is yet another helper script that takes care of putting the boiler plate application code in place, like the minimal React Native code in file index.ios.js and the Objective-C code plus other goodies in subfolder iOS, and the Xcode® project definition in subfolder AwesomeProject.xcodeproj

We already saw that the React Native based JavaScript code that makes up our actual application lives in `index.ios.js`.

The `package.json` file is no surprise for those who already worked with Node.js; it defines some metadata for our project and, most importantly, declares react-native (this time, the real thing, i.e., the actual framework that makes our application possible) as a dependency of our own project.

The `node_modules` folder is simply a result of the npm install run that took place during project initialization. It contains the react-native code, which in turn consists of other NPM dependencies, helper scripts, and a lot of JavaScript and Objective-C code.

The initialization process also provided the minimum Xcode project definitions plus some Objective-C boilerplate code, which allows us to open our new project in Xcode and to instantly run the application without any further ado. All of this could have been done manually, but would include a lot of steps that are identical no matter what kind of application we are going to write, thus it makes sense to streamline this process via react-native init.

To integrate thirth-party libraries into ReactNative project, please add the file Podfile under root folder.

`Podfile` is similar to `package.json`; it declares Objective-C library dependencies for the CocoaPods dependency manager. We will talk about this in more detail later in the book, and ignore it for now.

The takeaway here is that our AwesomeProject project is multiple things at once. It is an Xcode® project, but it's also an NPM project. It's a React Native based JavaScript application, but it also contains some iOS glue code that is needed to make our JavaScript code run on iOS in the first place.

Troubleshooting

- Install React Native Using CocoaPods

There are three use cases when you consider using `CocoaPods` in your projects - You want to integrate third-party libraries (that use `CocoaPods`) into your newly created ReactNative project. - Your project currently uses `CocoaPods` to manage dependencies and libraries. - You want to add to existing project the `ReactNative` framework.

`CocoaPods` is a package management tool for iOS/Mac development. We need to use it to download React Native. If you haven't install `CocoaPods` yet, Run this command to install.

```
gem install cocoapods
```

You will need to install `Ruby` to use the `gem` command

When you are ready to work with `CocoaPods`, add the following line to Podfile. If you don't have one, then create it under the root directory of your project.

```
pod 'React'
pod 'React/RCTText'
```

Add any subspecs you want to use in your project

Remember to install all subspecs you need. The

Then install your pods:

```
$ pod install
```

- Suppose you puts all `javascript` file under folder `ReactComponent`, and you use the `require` to import javascript module to use, consider add the file `package.json` under folder `React-Component`:

  package.json `{ "name": "my project", "version": "1.0.0" }`
- Start Development Server

In root directory, we need to start React Native development server.

```
(JS_DIR=`pwd`/ReactComponent; cd Pods/React; npm run start -- --root $JS_DIR)
```

This command will start up a React Native development server within our `CocoaPods` dependency to build our bundled script. The `--root` option indicates the root of your React Native apps – this will be our `ReactComponents` directory containing the single `index.ios.js` file. This running server will package up the index.ios.bundle file accessible via `http://localhost:8081/index.ios.bundle`

Then you can use the `require` function to import javascript modules.

- Linking Libraries and Modules

ReactNative now supports `cli` to generate new library:

```
react-native new-library SampleModule
```

After generating the `SampleModule`, you will need to link the `SampleModule.xcproject` to `XCode` by dragging into the `Libraries` folder under project name.

Remember to copy the `SampleModule.js` to the `ReactComponent` folder to use it.

In case your Xcode built failed. Please follow steps below:

- Clone main react native repository from github: `git clone https://github.com/facebook/react-native`
- Copy Examples/SampleApp to Examples/AwesomeProject `mkdir Examples/AwesomeProject cp -R Examples/SampleApp/* Examples/AwesomeProject`
- Run packager.sh to point to AwesomeProject just created: `packager/packager.sh --root=./Examples/Aweso`
- Now reopen again AwesomeProject in Xcode and start building.

Build and run on device:

Running On Device

Note that running on device requires Apple Developer account and provisioning your iPhone. This guide covers only React Native specific topic.

Accessing development server from device

You can iterate quickly on device using development server. To do that, your laptop and your phone have to be on the same wifi network.

```
    Open iOS/AppDelegate.m
    Change the IP in the URL from localhost to your laptop's IP
    In Xcode select your phone as build target and press "Build and run"
```

Using offline bundle

You can also pack all the JavaScript code within the app itself. This way you can test it without development server running and submit the app to the AppStore.

```
    Open iOS/AppDelegate.m
    Uncomment jsCodeLocation = [[NSBundle mainBundle] ...
    Run the react-native bundle command in terminal from the root directory of y\
our app
```

The bundle script supports a couple of flags:

```
    --dev - sets the value of __DEV__ variable to true. When true it turns on a \
bunch of useful development warnings. For production it is recommended to set __\
DEV__=false.
    --minify - pipe the JS code through UglifyJS.
```

If you started your project a while ago, main.jsbundle might not be included into Xcode project. To add it, right click on your project directory and click "Add Files to …" - choose the main.jsbundle file that you generated.

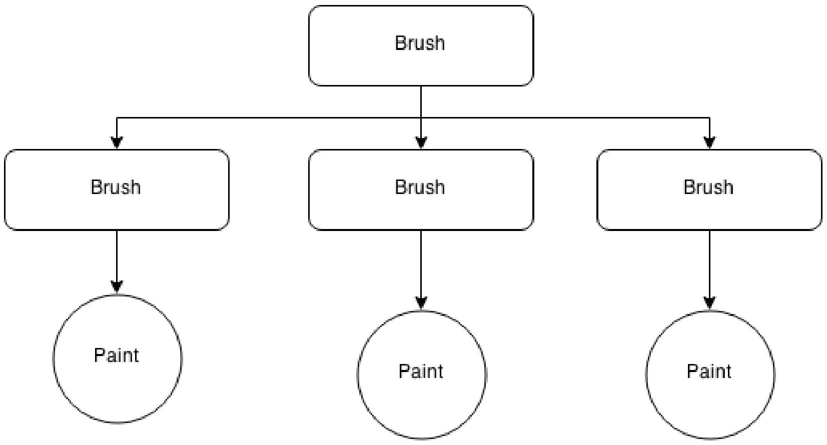
So what is React-Native ?

Phuhh. Congratulation if you come to this step. I hope you're seeing your AwesomeProject running on your iPhone 5. Now it's time to understand what's going on.

In traditional iOS architecture, if you're an artist, you have:

- A `Canvas`, it's the `UIWindow` class you often use to create your window application.
- A `Brush`, it's the `UIViewController` that you use to draw something on `Canvas`
- A `Paint`, it's the color and the view for your application. You use it with a `Brush`, and paint on a `Canvas`.

For more complex application design, you will have many `Brush`, as well as `Paint`. They're nested to others in a top-down manner to compose GUI.



ios

Example:

```
// Creating the Canvas
self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
// Creating the Brush
MyViewController  *rootViewController = [[MyViewController alloc] init];
rootViewController.launchOptions = launchOptions;
// Make the Canvas use the Brush
self.window.rootViewController = rootViewController;
[self.window makeKeyAndVisible];
return YES;
```

You create `Canvas` with `Frame`, then create the `Brush`, link it to the `Canvas` to make it the `rootViewController`. Inside the `Brush`, now you can create `Paint`, it's the `subView` of the `Brush`:

```
RCTRootView *rootView = [[RCTRootView alloc] initWithBundleURL:jsCodeLocation mo\
duleName:@"AdmobTest" launchOptions:self.launchOptions];
rootView.frame = CGRectMake(0, 50, viewRect.size.width, viewRect.size.height - 5\
0);
[self.view addSubview:rootView];
```

You can add as many `subView` as you want. Sweet!

So `React-Native` is just a `Paint`. If you see in the code below:

```
NSURL *jsCodeLocation;
jsCodeLocation = [[NSBundle mainBundle] URLForResource:@"main" withExtension:@"j\
sbundle"];
RCTRootView *rootView = [[RCTRootView alloc] initWithBundleURL:jsCodeLocation mo\
duleName:@"SampleApp" launchOptions:self.launchOptions];
```

you will see that to create new `React-Native` view, you have to pass two parameters: - The `jsLocation` address: it's the local or remote url for the `React component`. - The module name: It's the name that you register with `AppRegistry` in `index.ios.js` file:

```
`AppRegistry.registerComponent('SampleApp', () => SampleApp);`
```

After creating the `React-Native View`, now you can use its properties like any other `iOS View`, it's the `Paint` of your project.

Exercise:

It's enough for theory. There is a small exercise for you before you come to chapter 2.

```
Create new react native view, and put it inside a `ViewController`, style the vi\
ew with backgroundColor blue.
```

Solution:

Please do the exercise by yourself before coming here! If you're completely new to iOS, i recommend you to learn some basic Objective-C. There is a good resource[1] here for you to learn.

Now create two file `MyViewController.h` and `MyViewController.m` in xcode. The code for each file is listed below:

[1] http://learnxinyminutes.com/docs/objective-c/

```
// MyViewController.h
#import <UIKit/UIKit.h>
#import "RCTRootView.h"
@interface MyViewController : UIViewController
@property (weak, nonatomic) NSDictionary *launchOptions;
@end

// MyViewController.m
#import "MyViewController.h"

@implementation MyViewController
- (void)viewDidLoad
{
    [super viewDidLoad];
    NSURL *jsCodeLocation;
    jsCodeLocation = [[NSBundle mainBundle] URLForResource:@"main" withExtension\
:@"jsbundle"];

    RCTRootView *rootView = [[RCTRootView alloc] initWithBundleURL:jsCodeLocatio\
n moduleName:@"AdmobTest" launchOptions:self.launchOptions];
    rootView.frame = CGRectMake(0, 50, viewRect.size.width, viewRect.size.height\
 - 50);
    [self.view addSubview:rootView];
}
@end
```

Now modify the default `SampleApp's AppDelegate.m`:

```
#import "AppDelegate.h"
#import "MyViewController.h"
@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(\
NSDictionary *)launchOptions
{
  //NSURL *jsCodeLocation;

  // Loading JavaScript code - uncomment the one you want.

  // OPTION 1
  // Load from development server. Start the server from the repository root:
  //
  // $ npm start
  //
  // To run on device, change `localhost` to the IP address of your computer, an\
d make sure your computer and
  // iOS device are on the same Wi-Fi network.
  //jsCodeLocation = [NSURL URLWithString:@"http://localhost:8081/index.ios.bund\
le"];

  // OPTION 2
  // Load from pre-bundled file on disk. To re-generate the static bundle, run
  //
  // $ curl http://localhost:8081/index.ios.bundle -o main.jsbundle
  //
  // and uncomment the next following line


    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
    //UIViewController *rootViewController = [[UIViewController alloc] init];
    MyViewController  *rootViewController = [[MyViewController alloc] init];
    rootViewController.launchOptions = launchOptions;
    self.window.rootViewController = rootViewController;
    [self.window makeKeyAndVisible];
    return YES;
}

@end
```

You'll see i comment out all default `SampleApp` code, and put in our new MyViewController created above. Try to build and run project again. You will understand what's React Native View is about. We're ready to go to next step: Adding Flexbox layout.

Appendix: Quick introduction to React.js

React is, according to their own definition, "A Javascript library for building user interfaces." When I first read about react, it immediately struck me as a very good solution for a front end javascript framework. After making a couple small projects using react, I can conclude that it is a pretty great framework after all. In this post, I'll outline how to hit the ground running as quickly as possible with react. JSX — the syntax of React

React uses a specialized version of javascript called `jsx`. The basic idea is to provide some intuitive syntactic sugar to represent react's internal version of the DOM which is used by react components — the building blocks of a react-based application. React's official guide to jsx is quite comprehensive for learning the exact usage.

There are a few different ways to use `jsx` in your application, but at the end of the day jsx source code must eventually be converted to regular javascript. The first, and simplest, method is to simply include react's `JSXTransformer.js` file, which provides a way to convert jsx-labeled javascript into plain old javascript.

The second way, which I prefer, is to precompile jsx source code beforehand. This is most easily done by taking advantage of react's jsx command line tools:

```
$ npm install -g react-tools
$ cd project/
$ jsx --watch src/ build/
```

After doing this, using jsx is as simple as including `build/whatever.js` in your html file.

Getting up and running

After figuring out which jsx method you want to use, getting react working is trivial. Simply download `react.js` (or use a cdn) and include it in the head of your document:

```
<html>
    <head>
        <script src="react.js"></script>
        ...
    </head>
    <body>
        ...
        <script src="mine.js"></script>
    </body>
</html>
```

React doesn't depend on any external libraries (such as jquery), so you don't have to worry about that. However, make sure to include your source file after the content of your document.

Components — the building blocks of React

In react-land, Components are the central building blocks of your application. Components are self-contained, modular, dynamic representations of HTML in your application. Components are often children of other react Components. This makes more sense in context:

```
/** @jsx React.DOM */
var Thing = React.createClass({
  render: function() {
    return (
      <p>{this.props.name}</p>
    );
  }
});

var ThingList = React.createClass({
  render: function() {
    return (
      <h1>My Things:</h1>,
      <Thing name="Hello World!" />
    );
  }
});

React.renderComponent(
  <ThingList />,
  document.querySelector('body')
);
```

In this example, the `renderComponent` method kicks off the application by rendering a component into a given `DOM node`.

If this is your first time looking at `React`, a couple other things about this example should strike you as odd.

Is that HTML.. in my javascript?

No, not really. What you're seeing is `jsx` in action, but the lines with HTML-like syntax do not represent actual DOM nodes. Instead, they are syntactic sugar for react's internal representation of the DOM which also includes all components that you've previously declared. Code wrapped in curly braces is "escaped" from this syntax. Take a look at the precompiled version of the ThingList class:

```
var ThingList = React.createClass({displayName: 'ThingList',
  render: function() {
    return (
      React.DOM.h1(null, "My Things:"),
      Thing( {name:"Hello World!"} )
    );
  }
});
```

If this version makes more sense to you, that's completely expected. When I first started using `react`, I has a strong distaste for the HTML-like syntax. However, as your components become progressively more complex, you may start to appreciate `jsx` syntax more than you thought. For example:

```
var ComplexThing = React.createClass({
  render: function() {
    return (
      <div className="complexThing">
        <Thing name="thing one" />
        <Thing name="thing two" />
      </div>,
      <a href="back.html">Go Back</a>
    );
  }
});

// In comparison, the non-jsx version:

var ComplexThing = React.createClass({displayName: 'ComplexThing',
  render: function() {
    return (
      React.DOM.div( {className:"complexThing"},
        Thing( {name:"thing one"} ),
        Thing( {name:"thing two"} )
      ),
      React.DOM.a( {href:"back.html"}, "Go Back")
    );
  }
});
```

Components and State

In my original example you may have noticed something strange — `this.props.name`. Where did props come from? What does it do?

Every react component takes in `properties` - immutable information specific to a particular instance of that component. You can think of `this.props` as the parameters that get passed into a function. However, because they are immutable, there's no way to change the properties of a component after they are rendered. That's where state comes in.

In addition to immutable properties, react components have a private `this.state` attribute. When a state gets updated, the component re-renders itself.

```
/** @jsx React.DOM */
var StatefulThing = React.createClass({
  updateName: function(event) {
    event.preventDefault();
    this.state.name = "Taylor";
  },
  setInitialState: function() {
    return (
      {name: "World"}
    );
  },
  render: function() {
    return (
      <a href="#" onClick={this.updateName}>
        My name is {this.state.name}
      </a>
    );
  }
});
```

React attaches event handlers to components using a camelCase naming convention, and expects you to pass it a function to handle the event.

One thing to keep in mind about all of the examples that we've seen is that none of them make any assumptions about other parts of your application. React is purely a front end development framework. In contrast to frameworks such as `Backbone.js`, React expects you to do the work of getting and pushing data from the server. This makes it very easy to implement React as a front end solution, since it simply expects you to hand it data. React does all the other work.

Hopefully this introduction gives you an idea of how react works. If you want to learn more about how to use react, I recommend reading their tutorial, which gives a broad overview of React's features.