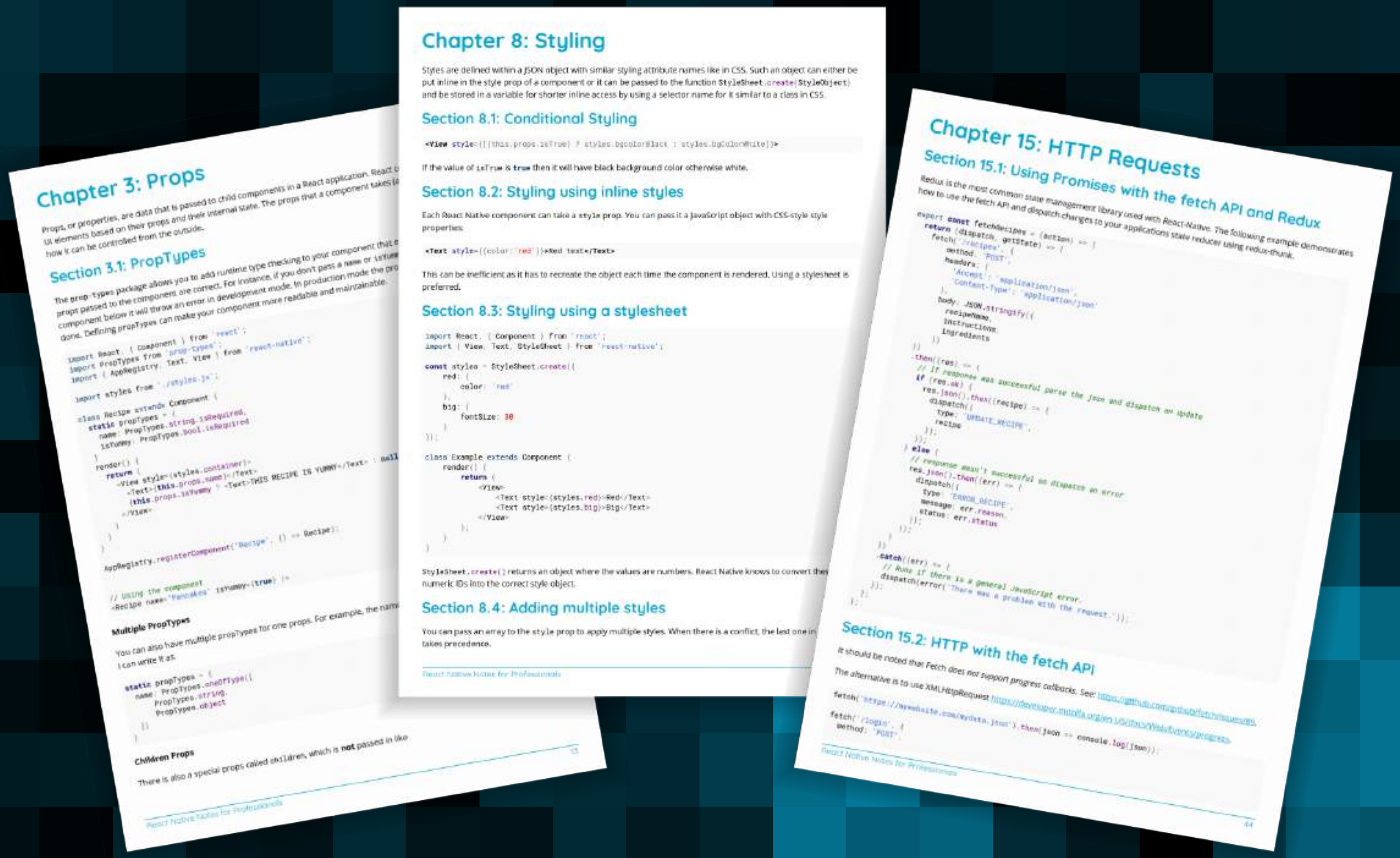
专业人员笔记

React Native

专业人员笔记

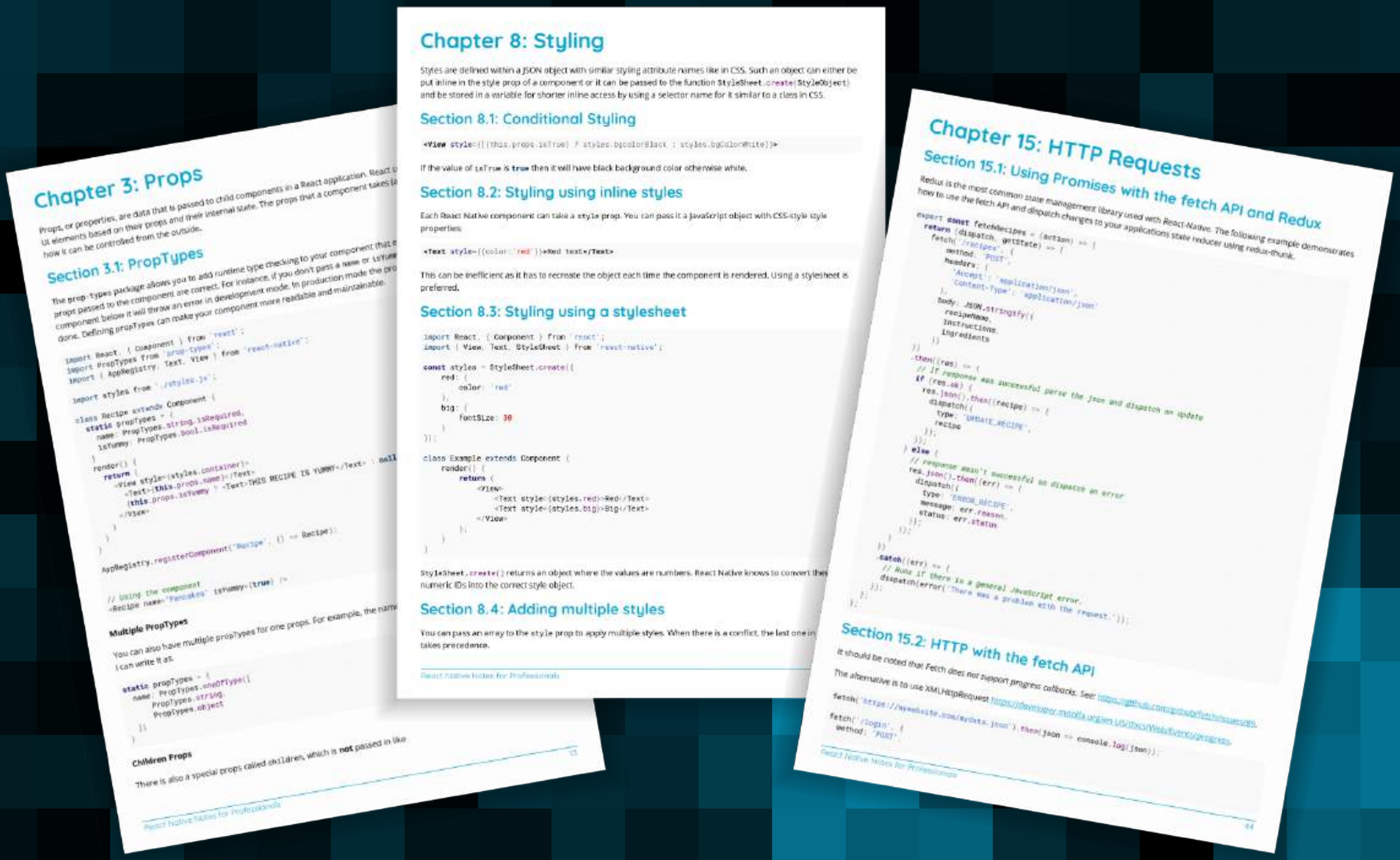


80+ 页

专业提示和技巧

React Native

Notes for Professionals



80+ pages

of professional hints and tricks

目录

Contents

关于

About

第1章：开始使用React Native

第1.1节：Mac环境设置

安装包管理器Homebrew `brew`

在终端提示符下粘贴该命令。

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

安装Xcode集成开发环境（IDE）

使用以下链接下载，或在Mac应用商店中查找

https://developer.apple.com/download/

> 注意：如果你同时安装了Xcode-beta.app和正式版的Xcode.app，确保你使用的是正式版的 xcodebuild工具。你可以通过以下命令设置：
>
> ```
> sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer/
> ```

安装Android环境

- Git `git`

  *如果您已经安装了XCode，Git已经安装，否则请运行以下命令

  ```
  brew install git
  ```

- 最新的JDK
- Android Studio

  选择自定义安装

Chapter 1: Getting started with React Native

Section 1.1: Setup for Mac

Installing package manager Homebrew `brew`

Paste that at a Terminal prompt.

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Installing Xcode IDE

Download it using link below or find it on Mac App Store

https://developer.apple.com/download/

> **NOTE:** If you have **Xcode-beta.app** installed along with production version of **Xcode.app**, make sure you are using production version of `xcodebuild` tool. You can set it with:
>
> ```
> sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer/
> ```

Installing Android environment

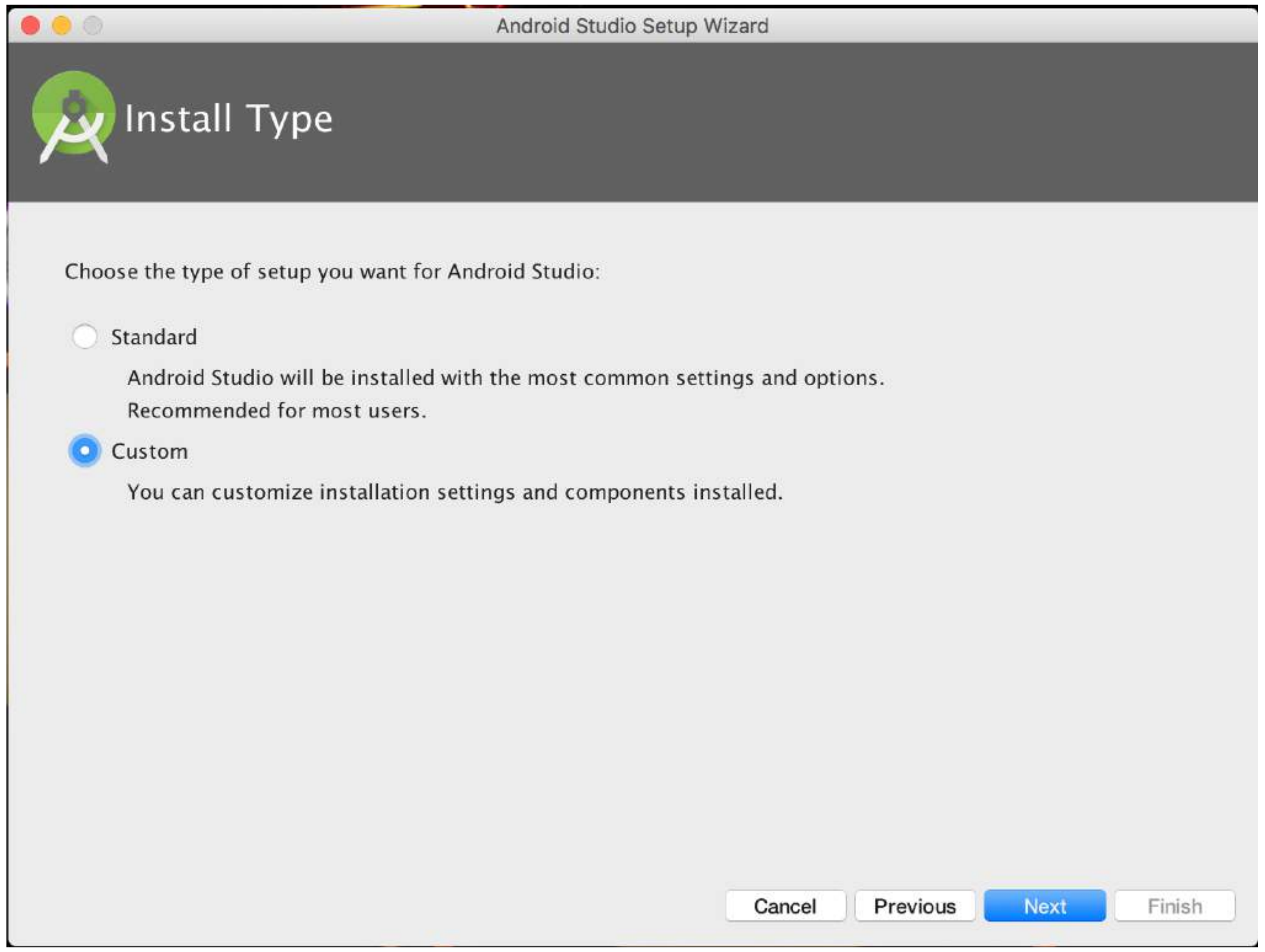
- Git `git`

  *If you have installed XCode, Git is already installed, otherwise run the following

  ```
  brew install git
  ```

- Latest JDK
- Android Studio

  Choose a Custom installation

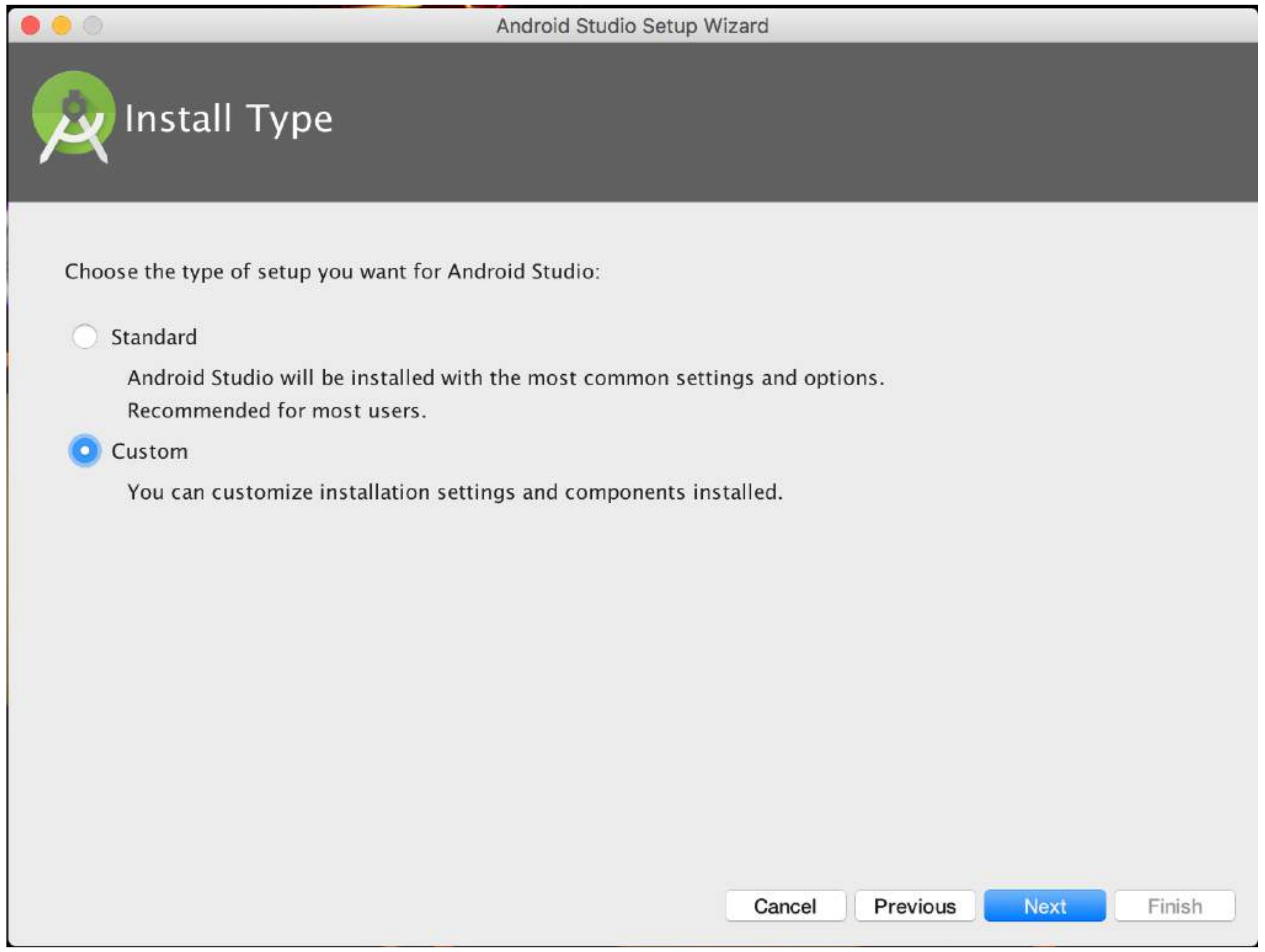


Choose the type of setup you want for Android Studio:

Standard
Android Studio will be installed with the most common settings and options.
Recommended for most users.

[✓ Custom]
You can customize installation settings and components installed.

选择性能和Android虚拟设备

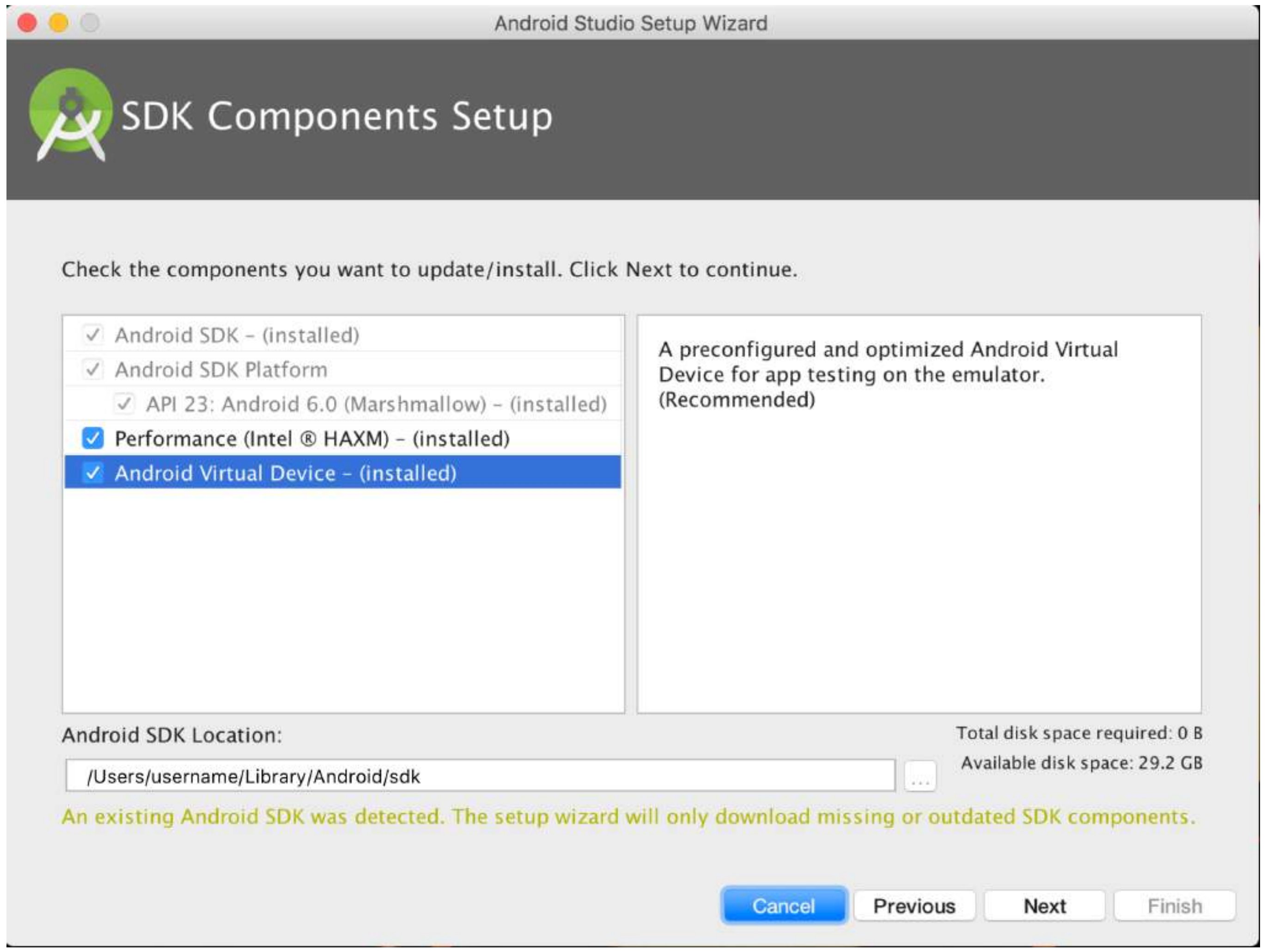


Choose the type of setup you want for Android Studio:

Standard
Android Studio will be installed with the most common settings and options.
Recommended for most users.

[✓ Custom]
You can customize installation settings and components installed.

Choose both Performance and Android Virtual Device



Android Studio Setup Wizard

SDK Components Setup

Check the components you want to update/install. Click Next to continue.

- [✓] Android SDK – (installed)
- [✓] Android SDK Platform
  - [✓] API 23: Android 6.0 (Marshmallow) – (installed)
- [✓] Performance (Intel ® HAXM) – (installed)
- [✓] Android Virtual Device – (installed)

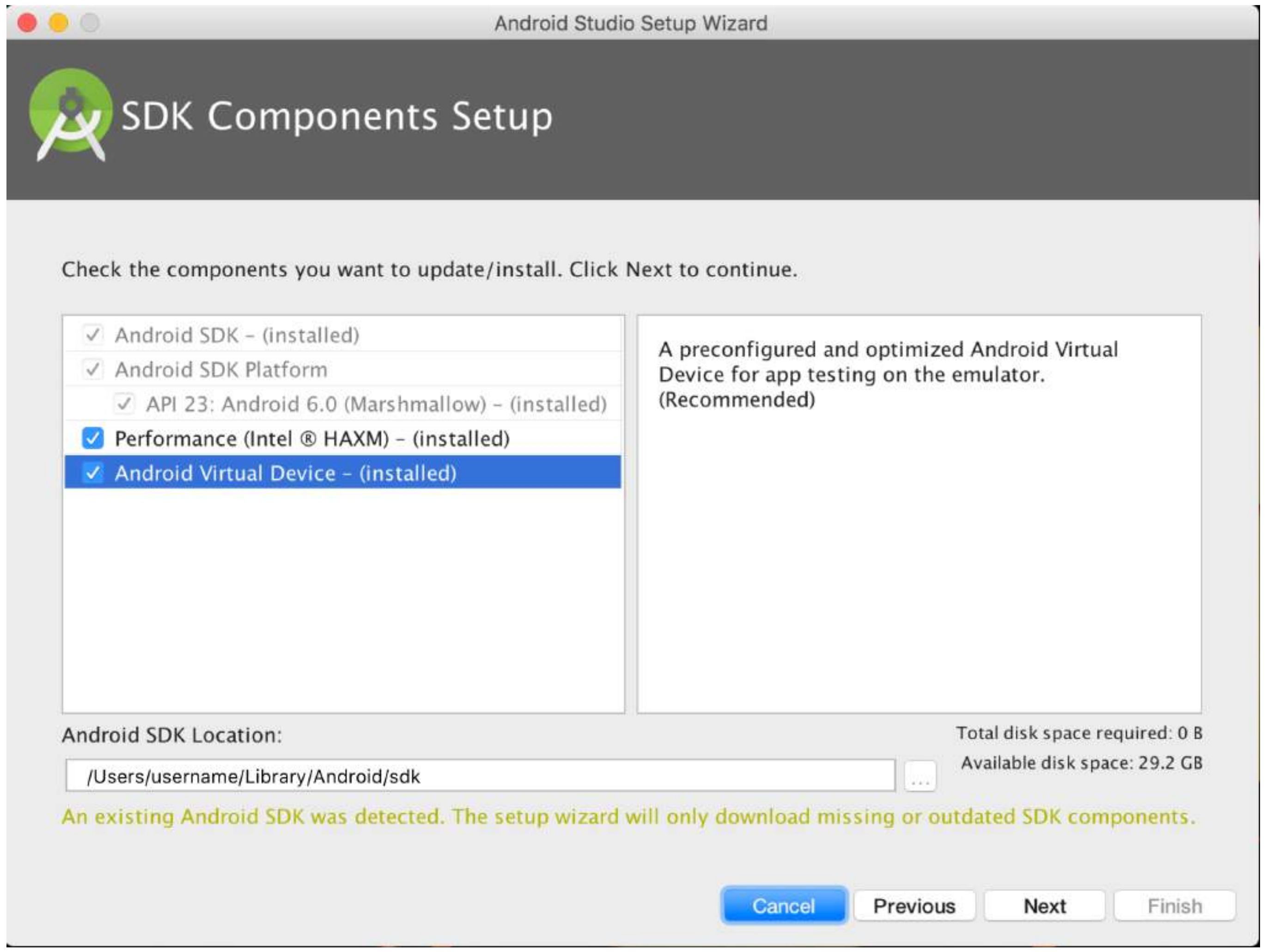
A preconfigured and optimized Android Virtual Device for app testing on the emulator. (Recommended)

Android SDK Location: /Users/username/Library/Android/sdk

Total disk space required: 0 B
Available disk space: 29.2 GB

An existing Android SDK was detected. The setup wizard will only download missing or outdated SDK components.

安装完成后，在Android Studio欢迎窗口选择配置 -> SDK管理器。



Android Studio Setup Wizard

SDK Components Setup

Check the components you want to update/install. Click Next to continue.

- [✓] Android SDK – (installed)
- [✓] Android SDK Platform
  - [✓] API 23: Android 6.0 (Marshmallow) – (installed)
- [✓] Performance (Intel ® HAXM) – (installed)
- [✓] Android Virtual Device – (installed)

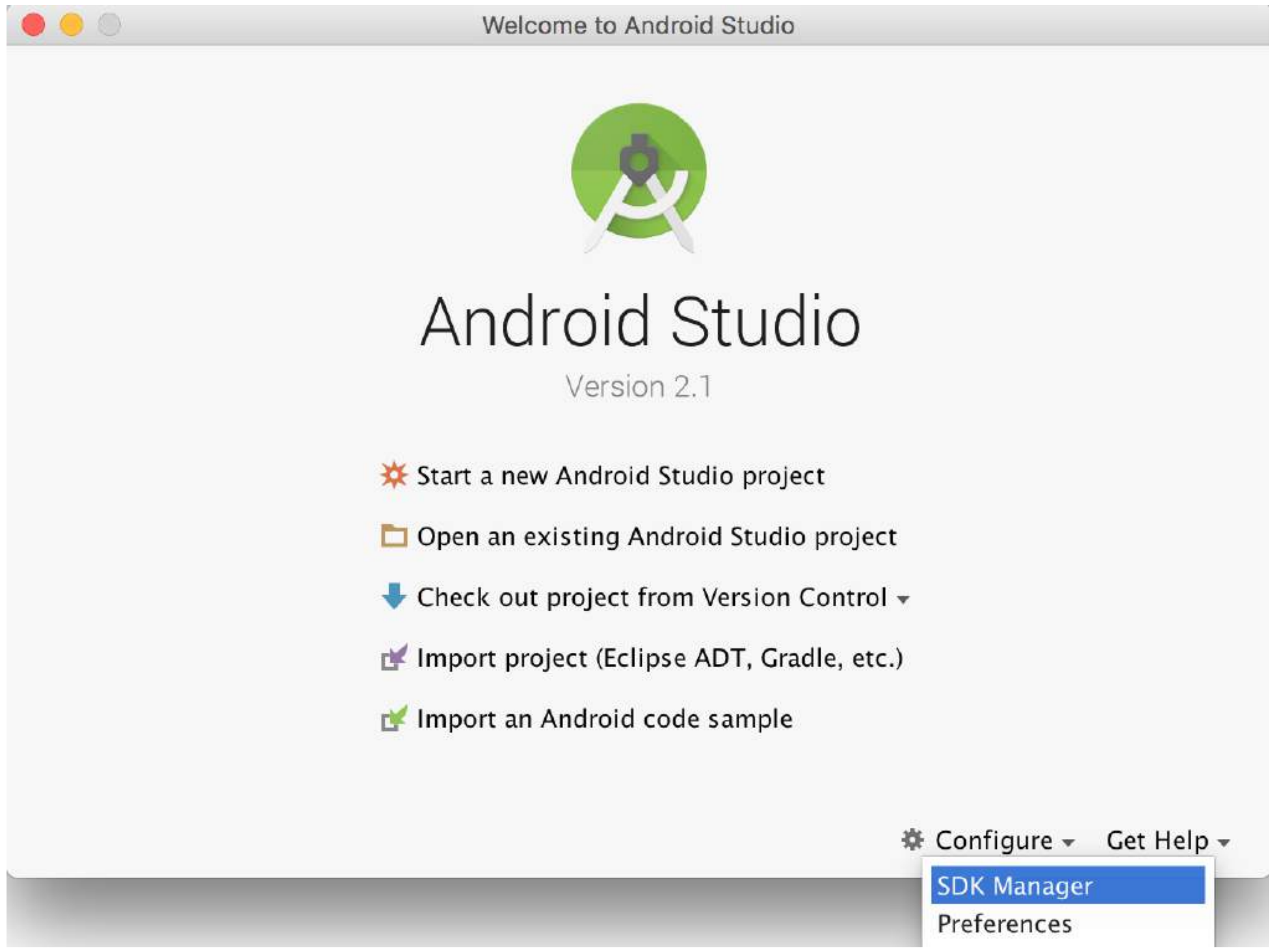
A preconfigured and optimized Android Virtual Device for app testing on the emulator. (Recommended)

Android SDK Location: /Users/username/Library/Android/sdk

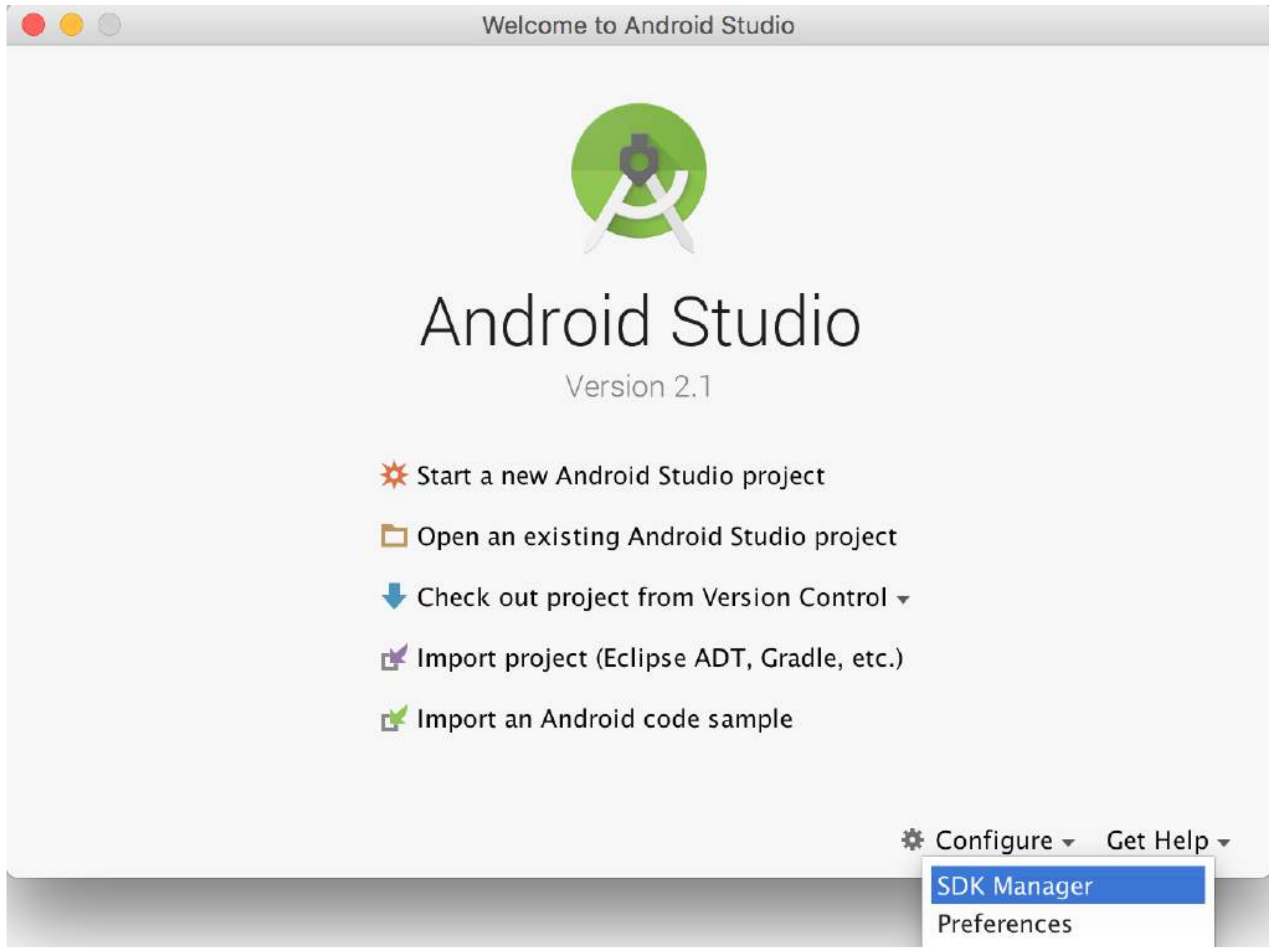
Total disk space required: 0 B
Available disk space: 29.2 GB

An existing Android SDK was detected. The setup wizard will only download missing or outdated SDK components.

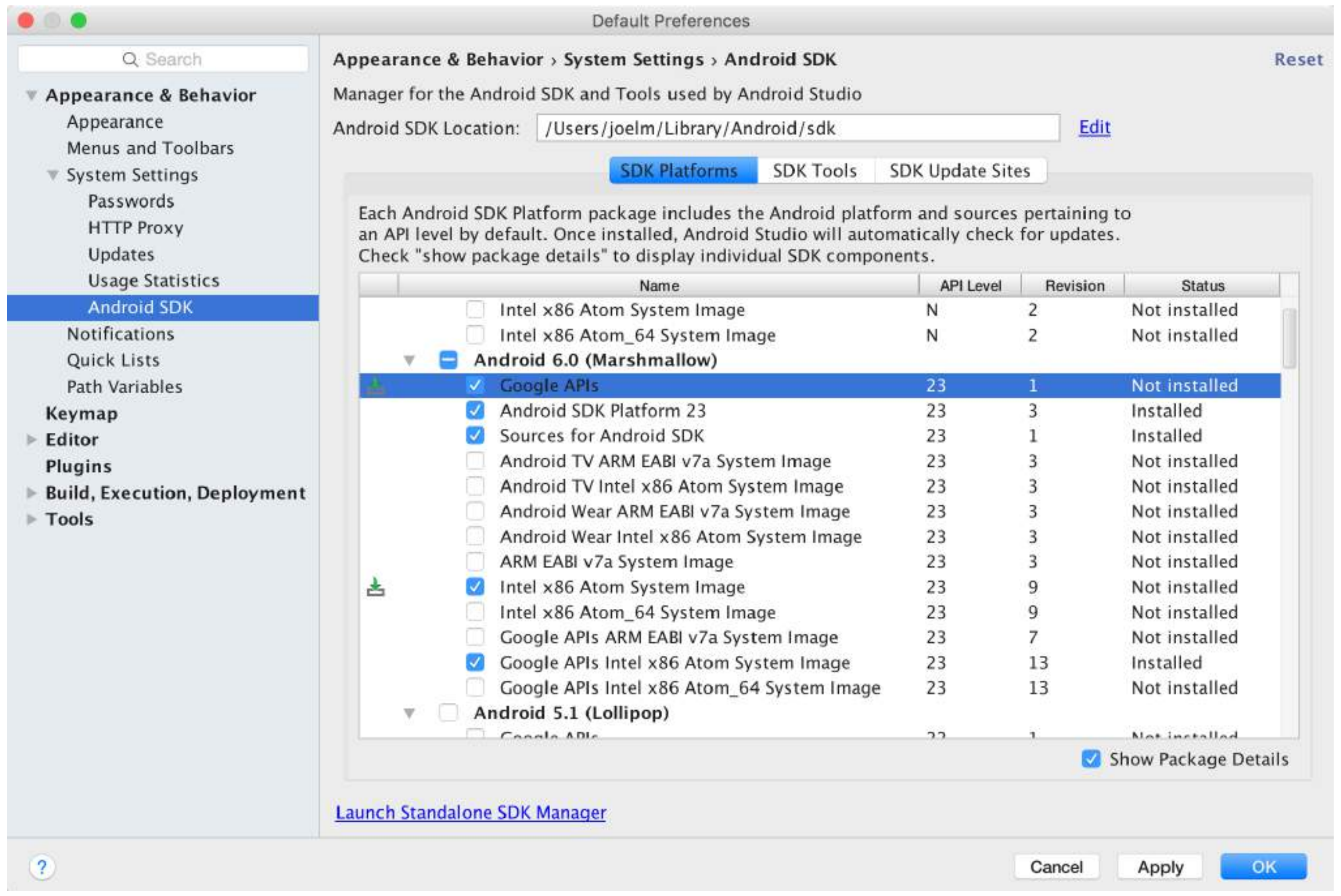
After installation, choose Configure -> SDK Manager from the Android Studio welcome window.


Welcome to Android Studio
Android Studio
Version 2.1
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
Configure
Get Help
SDK Manager
Preferences

在SDK平台窗口，选择显示包详情，并在Android 6.0（棉花糖）下，确保勾选了Google APIs、Intel x86 Atom系统镜像、Intel x86 Atom_64系统镜像以及Google APIs Intel x86 Atom_64系统镜像。


Welcome to Android Studio
Android Studio
Version 2.1
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
Configure
Get Help
SDK Manager
Preferences

In the SDK Platforms window, choose Show Package Details and under Android 6.0 (Marshmallow), make sure that Google APIs, Intel x86 Atom System Image, Intel x86 Atom_64 System Image, and Google APIs Intel x86 Atom_64 System Image are checked.



Default Preferences

Search

- Appearance & Behavior
  - Appearance
  - Menus and Toolbars
  - System Settings
    - Passwords
    - HTTP Proxy
    - Updates
    - Usage Statistics
    - Android SDK
  - Notifications
  - Quick Lists
  - Path Variables
- Keymap
- Editor
- Plugins
- Build, Execution, Deployment
- Tools

Appearance & Behavior › System Settings › Android SDK Reset

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/joelm/Library/Android/sdk Edit

SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

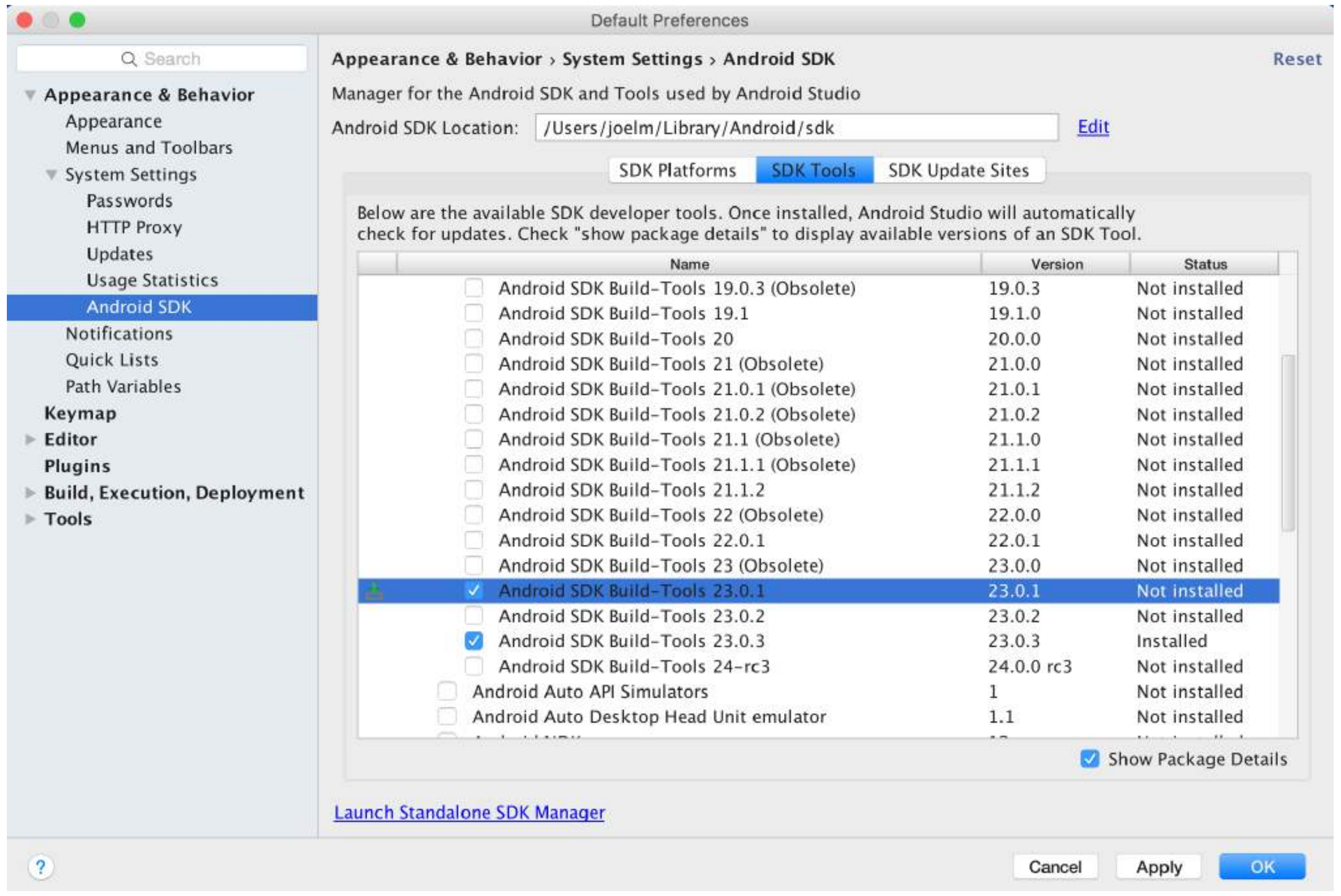
Name	API Level	Revision	Status
Intel x86 Atom System Image	N	2	Not installed
Intel x86 Atom_64 System Image	N	2	Not installed
Android 6.0 (Marshmallow)			
✓ Google APIs	23	1	Not installed
✓ Android SDK Platform 23	23	3	Installed
✓ Sources for Android SDK	23	1	Installed
Android TV ARM EABI v7a System Image	23	3	Not installed
Android TV Intel x86 Atom System Image	23	3	Not installed
Android Wear ARM EABI v7a System Image	23	3	Not installed
Android Wear Intel x86 Atom System Image	23	3	Not installed
ARM EABI v7a System Image	23	3	Not installed
✓ Intel x86 Atom System Image	23	9	Not installed
Intel x86 Atom_64 System Image	23	9	Not installed
Google APIs ARM EABI v7a System Image	23	7	Not installed
✓ Google APIs Intel x86 Atom System Image	23	13	Installed
Google APIs Intel x86 Atom_64 System Image	23	13	Not installed
Android 5.1 (Lollipop)			

✓ Show Package Details

Launch Standalone SDK Manager

Cancel Apply OK

在 SDK 工具窗口中，选择显示包详情，并确保在 Android SDK 构建工具下选中了 Android SDK Build-Tools 23.0.1。



Default Preferences

Search

- Appearance & Behavior
  - Appearance
  - Menus and Toolbars
  - System Settings
    - Passwords
    - HTTP Proxy
    - Updates
    - Usage Statistics
    - Android SDK
  - Notifications
  - Quick Lists
  - Path Variables
- Keymap
- Editor
- Plugins
- Build, Execution, Deployment
- Tools

Appearance & Behavior › System Settings › Android SDK Reset

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/joelm/Library/Android/sdk Edit

SDK Platforms | SDK Tools | SDK Update Sites

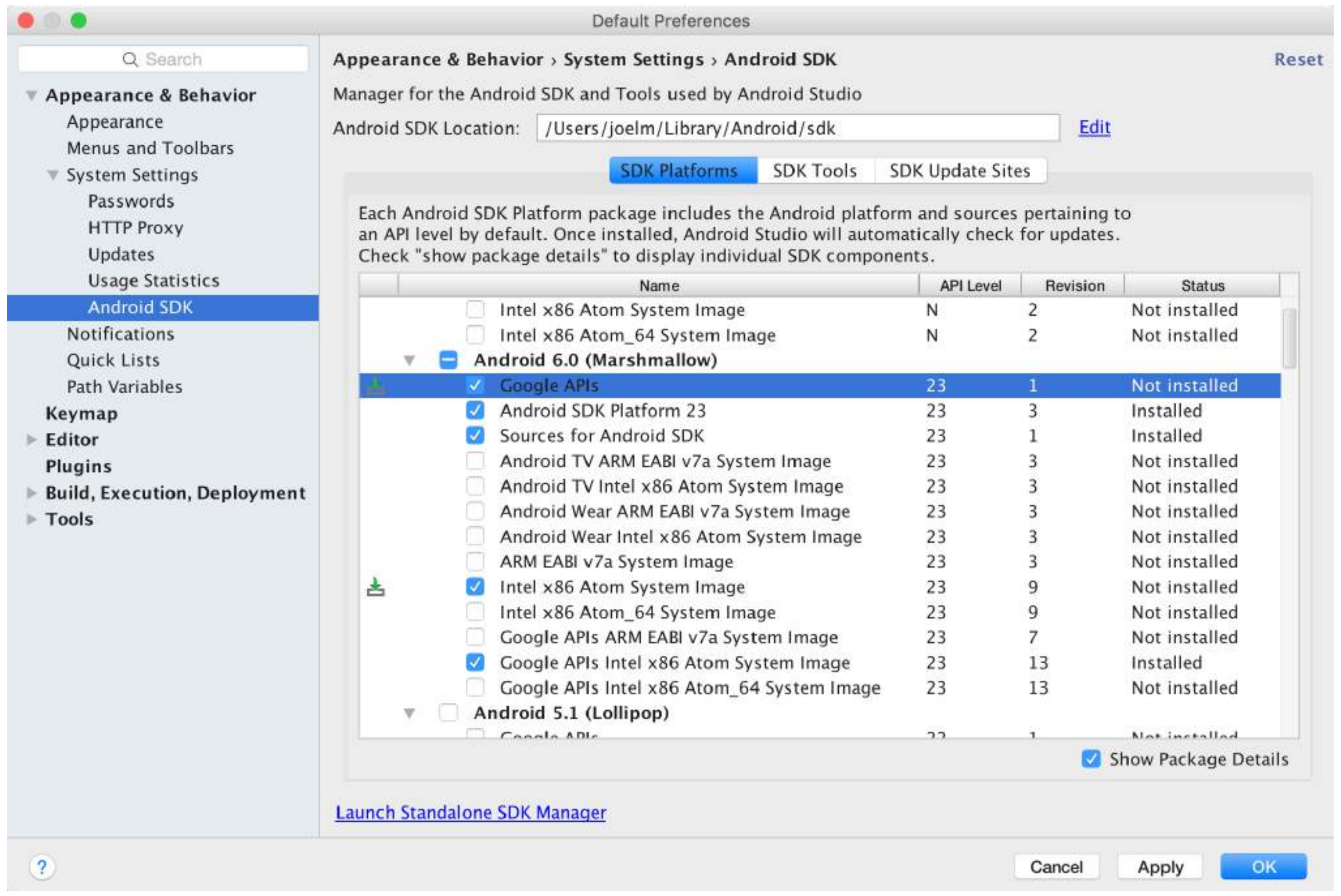
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
Android SDK Build-Tools 19.0.3 (Obsolete)	19.0.3	Not installed
Android SDK Build-Tools 19.1	19.1.0	Not installed
Android SDK Build-Tools 20	20.0.0	Not installed
Android SDK Build-Tools 21 (Obsolete)	21.0.0	Not installed
Android SDK Build-Tools 21.0.1 (Obsolete)	21.0.1	Not installed
Android SDK Build-Tools 21.0.2 (Obsolete)	21.0.2	Not installed
Android SDK Build-Tools 21.1 (Obsolete)	21.1.0	Not installed
Android SDK Build-Tools 21.1.1 (Obsolete)	21.1.1	Not installed
Android SDK Build-Tools 21.1.2	21.1.2	Not installed
Android SDK Build-Tools 22 (Obsolete)	22.0.0	Not installed
Android SDK Build-Tools 22.0.1	22.0.1	Not installed
Android SDK Build-Tools 23 (Obsolete)	23.0.0	Not installed
✓ Android SDK Build-Tools 23.0.1	23.0.1	Not installed
Android SDK Build-Tools 23.0.2	23.0.2	Not installed
✓ Android SDK Build-Tools 23.0.3	23.0.3	Installed
Android SDK Build-Tools 24-rc3	24.0.0 rc3	Not installed
Android Auto API Simulators	1	Not installed
Android Auto Desktop Head Unit emulator	1.1	Not installed

✓ Show Package Details

Launch Standalone SDK Manager

Cancel Apply OK



Default Preferences

Search

- Appearance & Behavior
  - Appearance
  - Menus and Toolbars
  - System Settings
    - Passwords
    - HTTP Proxy
    - Updates
    - Usage Statistics
    - Android SDK
  - Notifications
  - Quick Lists
  - Path Variables
- Keymap
- Editor
- Plugins
- Build, Execution, Deployment
- Tools

Appearance & Behavior › System Settings › Android SDK Reset

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/joelm/Library/Android/sdk Edit

SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

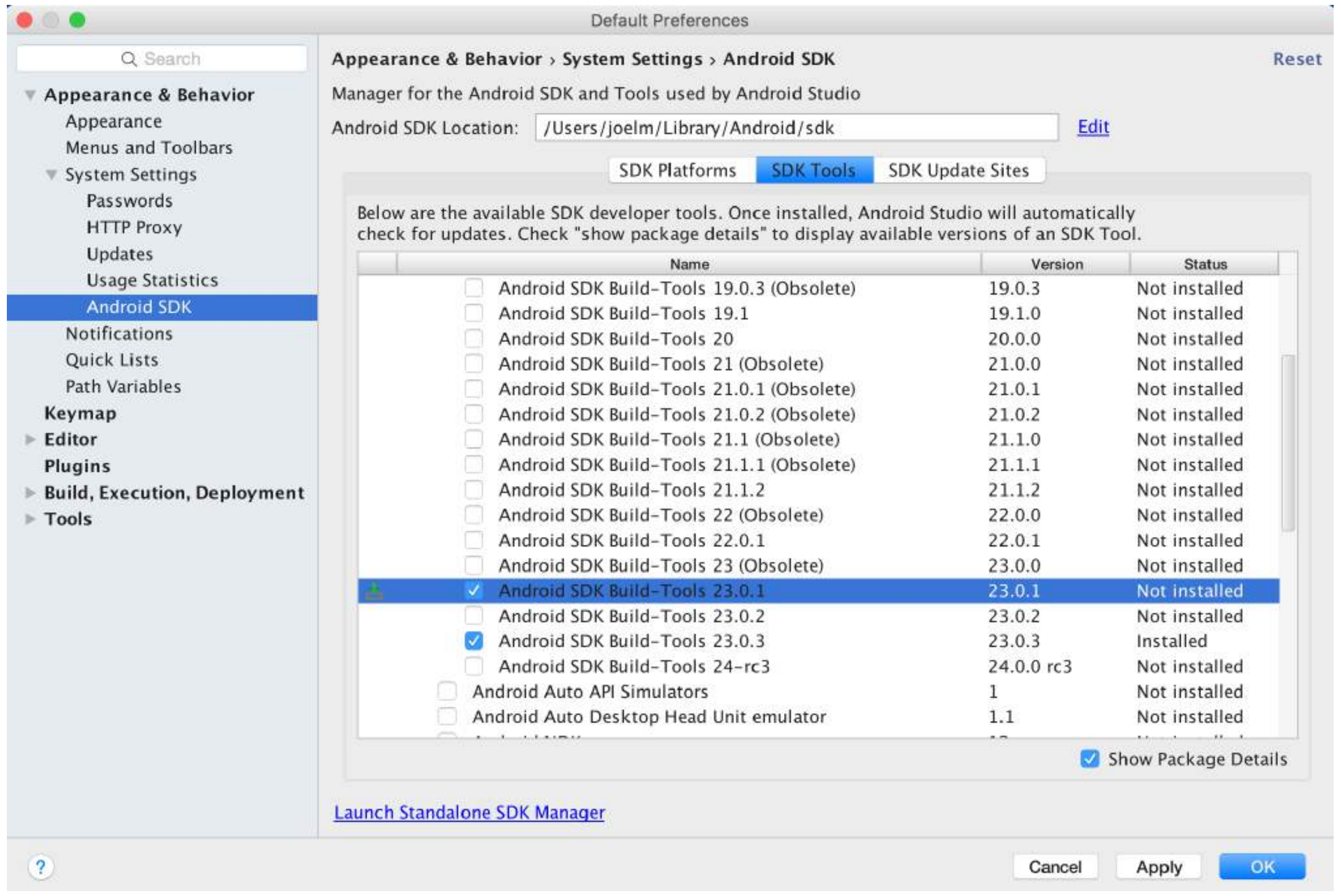
Name	API Level	Revision	Status
Intel x86 Atom System Image	N	2	Not installed
Intel x86 Atom_64 System Image	N	2	Not installed
Android 6.0 (Marshmallow)			
✓ Google APIs	23	1	Not installed
✓ Android SDK Platform 23	23	3	Installed
✓ Sources for Android SDK	23	1	Installed
Android TV ARM EABI v7a System Image	23	3	Not installed
Android TV Intel x86 Atom System Image	23	3	Not installed
Android Wear ARM EABI v7a System Image	23	3	Not installed
Android Wear Intel x86 Atom System Image	23	3	Not installed
ARM EABI v7a System Image	23	3	Not installed
✓ Intel x86 Atom System Image	23	9	Not installed
Intel x86 Atom_64 System Image	23	9	Not installed
Google APIs ARM EABI v7a System Image	23	7	Not installed
✓ Google APIs Intel x86 Atom System Image	23	13	Installed
Google APIs Intel x86 Atom_64 System Image	23	13	Not installed
Android 5.1 (Lollipop)			

✓ Show Package Details

Launch Standalone SDK Manager

Cancel Apply OK

In the SDK Tools window, choose Show Package Details and under Android SDK Build Tools, make sure that Android SDK Build-Tools 23.0.1 is selected.



Default Preferences

Search

- Appearance & Behavior
  - Appearance
  - Menus and Toolbars
  - System Settings
    - Passwords
    - HTTP Proxy
    - Updates
    - Usage Statistics
    - Android SDK
  - Notifications
  - Quick Lists
  - Path Variables
- Keymap
- Editor
- Plugins
- Build, Execution, Deployment
- Tools

Appearance & Behavior › System Settings › Android SDK Reset

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/joelm/Library/Android/sdk Edit

SDK Platforms | SDK Tools | SDK Update Sites

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
Android SDK Build-Tools 19.0.3 (Obsolete)	19.0.3	Not installed
Android SDK Build-Tools 19.1	19.1.0	Not installed
Android SDK Build-Tools 20	20.0.0	Not installed
Android SDK Build-Tools 21 (Obsolete)	21.0.0	Not installed
Android SDK Build-Tools 21.0.1 (Obsolete)	21.0.1	Not installed
Android SDK Build-Tools 21.0.2 (Obsolete)	21.0.2	Not installed
Android SDK Build-Tools 21.1 (Obsolete)	21.1.0	Not installed
Android SDK Build-Tools 21.1.1 (Obsolete)	21.1.1	Not installed
Android SDK Build-Tools 21.1.2	21.1.2	Not installed
Android SDK Build-Tools 22 (Obsolete)	22.0.0	Not installed
Android SDK Build-Tools 22.0.1	22.0.1	Not installed
Android SDK Build-Tools 23 (Obsolete)	23.0.0	Not installed
✓ Android SDK Build-Tools 23.0.1	23.0.1	Not installed
Android SDK Build-Tools 23.0.2	23.0.2	Not installed
✓ Android SDK Build-Tools 23.0.3	23.0.3	Installed
Android SDK Build-Tools 24-rc3	24.0.0 rc3	Not installed
Android Auto API Simulators	1	Not installed
Android Auto Desktop Head Unit emulator	1.1	Not installed

✓ Show Package Details

Launch Standalone SDK Manager

Cancel Apply OK

- 环境变量 ANDROID_HOME

  确保 ANDROID_HOME 环境变量指向你现有的 Android SDK。为此，将以下内容添加到你的 ~/.bashrc、~/.bash_profile（或你所使用的 shell 配置文件）中，然后重新打开终端：

  如果你是在没有安装 Android Studio 的情况下安装 SDK，那么路径可能类似于：/usr/local/opt/android-sdk

  ```
  export ANDROID_HOME=~/Library/Android/sdk
  ```

Mac 依赖项

你需要为 iOS 安装 Xcode，为 Android 安装 Android Studio，还需要 node.js、React Native 命令行工具和 Watchman。

我们建议通过 Homebrew 安装 node 和 watchman。

```
brew install node
brew install watchman
```

> Watchman 是 Facebook 提供的一个用于监控文件系统变化的工具。强烈建议你安装它以获得更好的性能。它是可选的。

Node 自带 npm，npm 允许你安装 React Native 命令行界面。

```
npm install -g react-native-cli
```

如果遇到权限错误，尝试使用 sudo：

```
sudo npm install -g react-native-cli。
```

对于 iOS，安装 Xcode 最简单的方式是通过 Mac App Store。对于 Android，请下载并安装 Android Studio。

如果你计划修改 Java 代码，建议使用 Gradle Daemon，它可以加快构建速度。

测试你的 React Native 安装

使用 React Native 命令行工具生成一个名为“AwesomeProject”的新 React Native 项目，然后在新创建的文件夹内运行 react-native run-ios。

```
react-native init AwesomeProject
cd AwesomeProject
react-native run-ios
```

您应该很快会在 iOS 模拟器中看到您的新应用运行。react-native run-ios 只是运行您的应用的一种方式——您也可以直接从 Xcode 或 Nuclide 中运行它。

修改您的应用

既然您已经成功运行了应用，让我们来修改它。

- 在您选择的文本编辑器中打开 index.ios.js 或 index.android.js 并编辑几行代码。

- Environment Variable ANDROID_HOME

  Ensure the ANDROID_HOME environment variable points to your existing Android SDK. To do that, add this to your ~/.bashrc, ~/.bash_profile (or whatever your shell uses) and re-open your terminal:

  If you installed the SDK without Android Studio, then it may be something like: /usr/local/opt/android-sdk

  ```
  export ANDROID_HOME=~/Library/Android/sdk
  ```

Dependencies for Mac

You will need Xcode for iOS and Android Studio for android, node.js, the React Native command line tools, and Watchman.

We recommend installing node and watchman via Homebrew.

```
brew install node
brew install watchman
```

> Watchman is a tool by Facebook for watching changes in the filesystem. It is highly recommended you install it for better performance. It is optional.

Node comes with npm, which lets you install the React Native command line interface.

```
npm install -g react-native-cli
```

If you get a permission error, try with sudo:

```
sudo npm install -g react-native-cli.
```

For iOS the easiest way to install Xcode is via the Mac App Store. And for android download and install Android Studio.

If you plan to make changes in Java code, we recommend Gradle Daemon which speeds up the build.

Testing your React Native Installation

Use the React Native command line tools to generate a new React Native project called "AwesomeProject", then run react-native run-ios inside the newly created folder.

```
react-native init AwesomeProject
cd AwesomeProject
react-native run-ios
```

You should see your new app running in the iOS Simulator shortly. react-native run-ios is just one way to run your app - you can also run it directly from within Xcode or Nuclide.

Modifying your app

Now that you have successfully run the app, let's modify it.

- Open index.ios.js or index.android.js in your text editor of choice and edit some lines.

- 在您的 iOS 模拟器中按 Command⌘ + R 重新加载应用并查看您的更改！就是这么简单！

恭喜！您已成功运行并修改了您的第一个 React Native 应用。

来源：[Getting Started - React-Native](#)

第 1.2 节：Linux（Ubuntu）设置

1）安装 Node.JS

打开终端并运行以下命令以安装 nodeJS：

```
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash -

sudo apt-get install nodejs
```

如果 node 命令不可用

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

NodeJS 的其他安装方式：

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

或者

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install -y nodejs
```

检查你是否拥有当前版本

```
node -v
```

运行 npm 来安装 react-native

```
sudo npm install -g react-native-cli
```

2) 安装 Java

```
sudo apt-get install lib32stdc++6 lib32z1 openjdk-7-jdk
```

3) 安装 Android Studio：

Android SDK 或 Android Studio

```
http://developer.android.com/sdk/index.html
```

Android SDK 和 环境变量

```
export ANDROID_HOME=/YOUR/LOCAL/ANDROID/SDK
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

4) 设置模拟器：

在终端运行命令

```
android
```

在 SDK 管理器中选择“SDK Platforms”，你应该会看到“Android 7.0（Nougat）”旁边有一个蓝色的勾选标记

- Hit Command⌘ + R in your iOS Simulator to reload the app and see your change! That's it!

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

source: [Getting Started - React-Native](#)

Section 1.2: Setup for Linux (Ubuntu)

1) Setup Node.JS

Start the terminal and run the following commands to install nodeJS:

```
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash -

sudo apt-get install nodejs
```

If node command is unavailable

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Alternatives NodeJS instalations:

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

or

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install -y nodejs
```

check if you have the current version

```
node -v
```

Run the npm to install the react-native

```
sudo npm install -g react-native-cli
```

2) Setup Java

```
sudo apt-get install lib32stdc++6 lib32z1 openjdk-7-jdk
```

3) Setup Android Studio:

Android SDK or Android Studio

```
http://developer.android.com/sdk/index.html
```

Android SDK e ENV

```
export ANDROID_HOME=/YOUR/LOCAL/ANDROID/SDK
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

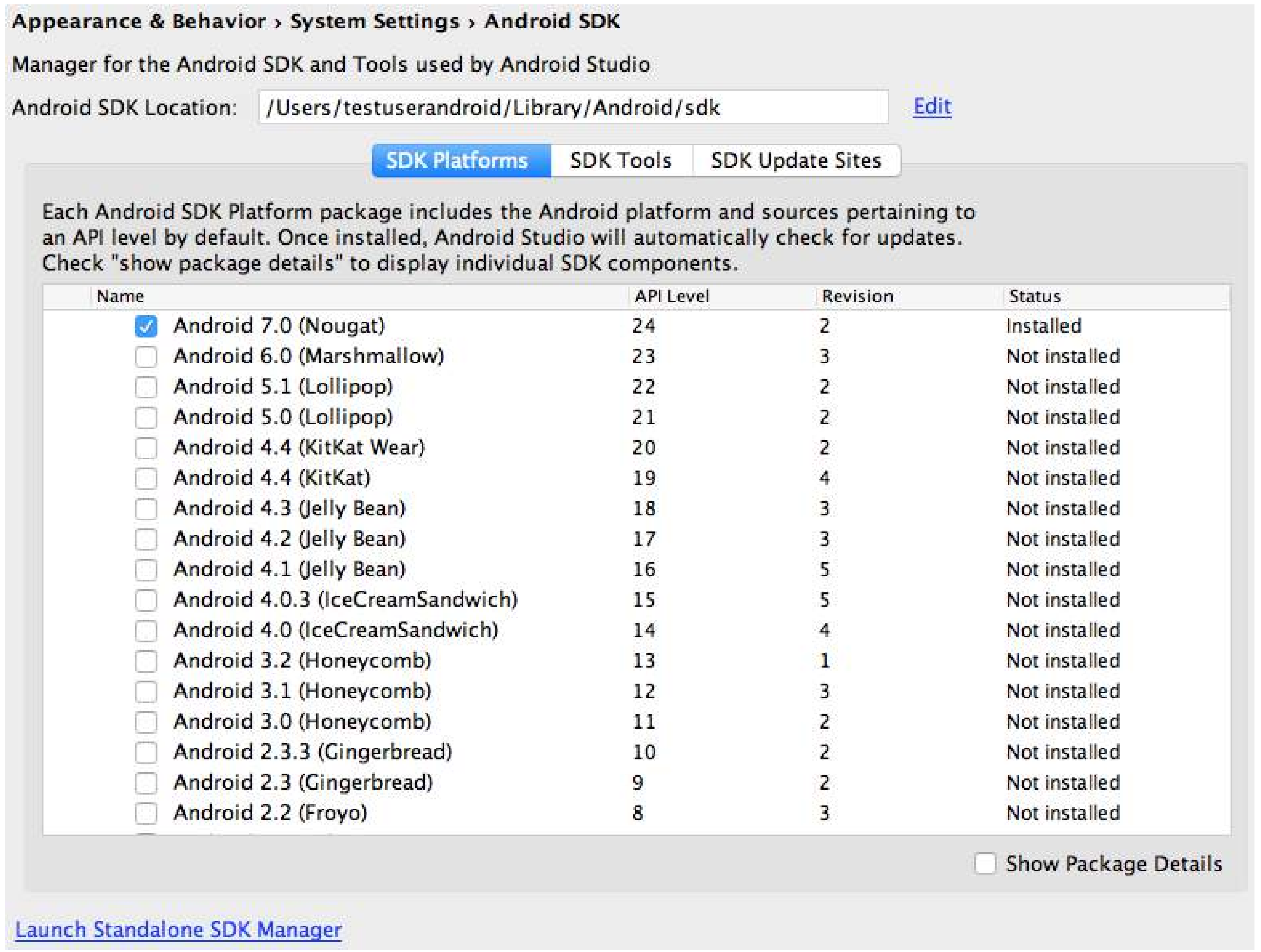
4) Setup emulator:

On the terminal run the command

```
android
```

Select "SDK Platforms" from within the SDK Manager and you should see a blue checkmark next to "Android 7.0

（Nougat）”。如果没有，点击复选框然后点击“应用”。



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/testuserandroid/Library/Android/sdk Edit

SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
☑ Android 7.0 (Nougat)	24	2	Installed
☐ Android 6.0 (Marshmallow)	23	3	Not installed
☐ Android 5.1 (Lollipop)	22	2	Not installed
☐ Android 5.0 (Lollipop)	21	2	Not installed
☐ Android 4.4 (KitKat Wear)	20	2	Not installed
☐ Android 4.4 (KitKat)	19	4	Not installed
☐ Android 4.3 (Jelly Bean)	18	3	Not installed
☐ Android 4.2 (Jelly Bean)	17	3	Not installed
☐ Android 4.1 (Jelly Bean)	16	5	Not installed
☐ Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
☐ Android 4.0 (IceCreamSandwich)	14	4	Not installed
☐ Android 3.2 (Honeycomb)	13	1	Not installed
☐ Android 3.1 (Honeycomb)	12	3	Not installed
☐ Android 3.0 (Honeycomb)	11	2	Not installed
☐ Android 2.3.3 (Gingerbread)	10	2	Not installed
☐ Android 2.3 (Gingerbread)	9	2	Not installed
☐ Android 2.2 (Froyo)	8	3	Not installed

☐ Show Package Details

Launch Standalone SDK Manager

5）启动一个项目

示例应用初始化

```
react-native init ReactNativeDemo && cd ReactNativeDemo
```

注意：始终检查android/app/build.gradle中的版本是否与您在安卓 SDK 中下载的构建工具版本相同

```
android {
compileSdkVersion XX
    buildToolsVersion "XX.X.X"
...
```

6）运行项目

打开 Android AVD 以设置虚拟安卓设备。执行命令行：

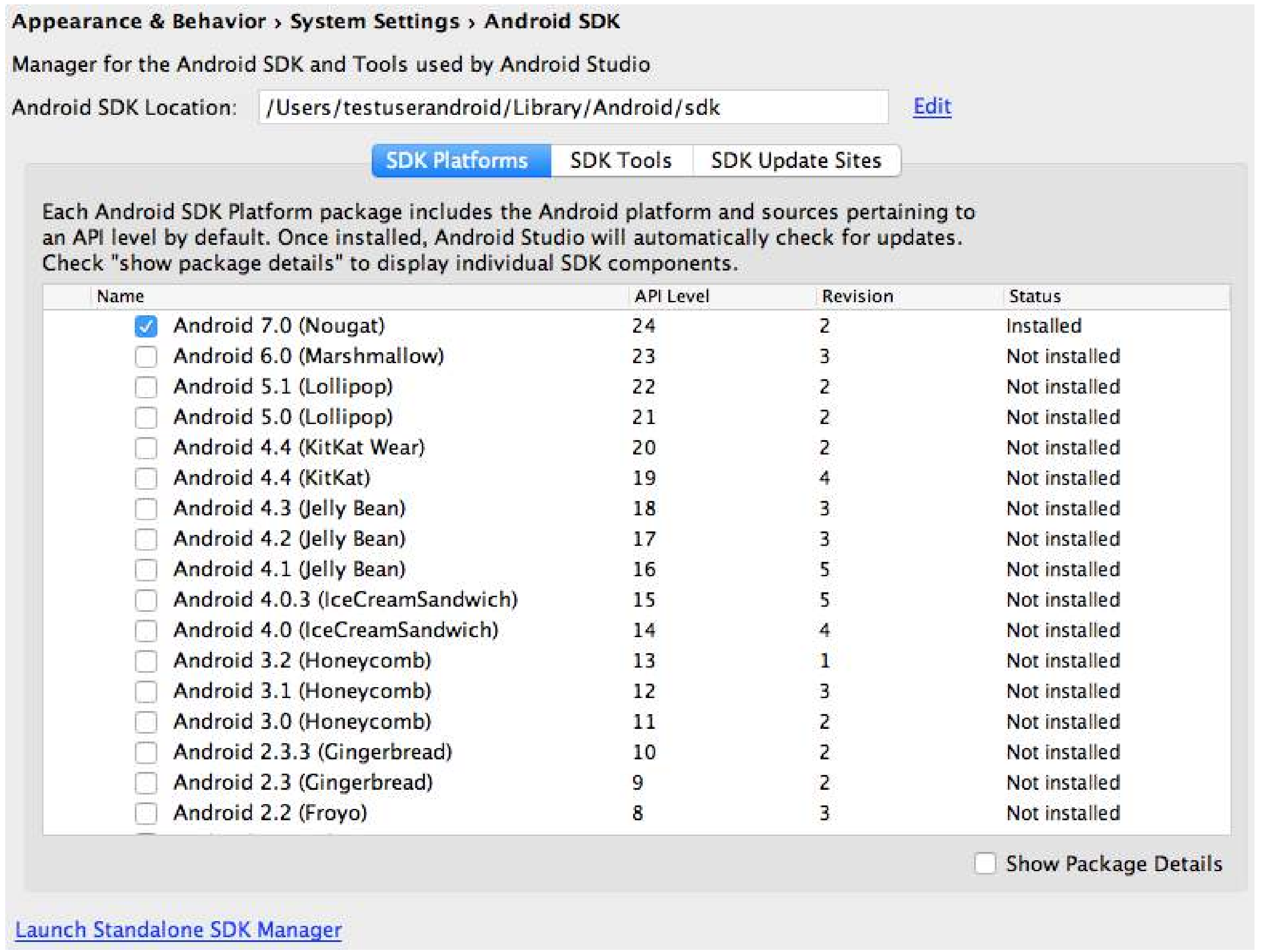
```
安卓模拟器
```

按照说明创建虚拟设备并启动它

打开另一个终端并运行以下命令行：

```
react-native run-android
```

(Nougat)". In case it is not, click on the checkbox and then "Apply".



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/testuserandroid/Library/Android/sdk Edit

SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
☑ Android 7.0 (Nougat)	24	2	Installed
☐ Android 6.0 (Marshmallow)	23	3	Not installed
☐ Android 5.1 (Lollipop)	22	2	Not installed
☐ Android 5.0 (Lollipop)	21	2	Not installed
☐ Android 4.4 (KitKat Wear)	20	2	Not installed
☐ Android 4.4 (KitKat)	19	4	Not installed
☐ Android 4.3 (Jelly Bean)	18	3	Not installed
☐ Android 4.2 (Jelly Bean)	17	3	Not installed
☐ Android 4.1 (Jelly Bean)	16	5	Not installed
☐ Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
☐ Android 4.0 (IceCreamSandwich)	14	4	Not installed
☐ Android 3.2 (Honeycomb)	13	1	Not installed
☐ Android 3.1 (Honeycomb)	12	3	Not installed
☐ Android 3.0 (Honeycomb)	11	2	Not installed
☐ Android 2.3.3 (Gingerbread)	10	2	Not installed
☐ Android 2.3 (Gingerbread)	9	2	Not installed
☐ Android 2.2 (Froyo)	8	3	Not installed

☐ Show Package Details

Launch Standalone SDK Manager

5) Start a project

Example app init

```
react-native init ReactNativeDemo && cd ReactNativeDemo
```

Obs: Always check if the version on `android/app/build.gradle` is the same as the Build Tools downloaded on your android SDK

```
android {
    compileSdkVersion XX
    buildToolsVersion "XX.X.X"
...
```

6) Run the project

Open Android AVD to set up a virtual android. Execute the command line:

```
android avd
```

Follow the instructions to create a virtual device and start it

Open another terminal and run the command lines:

```
react-native run-android
```

```
react-native start
```

第1.3节：Windows设置

注意：您无法在Windows上开发iOS的react-native应用，只能开发react-native安卓应用。

react-native在Windows上的官方设置文档可以在这里找到。如果需要更详细的信息，这里有一个详细的指南。

工具/环境

- Windows 10
- 命令行工具（例如PowerShell或Windows命令行）
- Chocolatey（通过PowerShell设置的步骤）
- JDK（版本8）
- Android Studio
- 一台启用了虚拟化技术以支持HAXM的英特尔机器（可选，仅当你想使用模拟器时需要）

1）为React Native开发设置你的机器

以管理员身份启动命令行，运行以下命令：

```
choco install nodejs.install
choco install python2
```

重新以管理员身份启动命令行，以便你可以运行npm

```
npm install -g react-native-cli
```

运行最后一个命令后，复制react-native安装的目录。你将在第4步中需要它。我在两台电脑上尝试过，一台是：C:\Program Files (x86)\Nodist\v-x64\6.2.2。另一台是：C:\Users\admin\AppData\Roamingpm

2）设置你的环境变量

本节的图文分步指南可在此处找到。

通过导航打开环境变量窗口：

[右键点击] “开始”菜单 -> 系统 -> 高级系统设置 -> 环境变量

在底部区域找到“Path”系统变量，并添加在步骤中安装 react-native 的位置
1.

如果你还没有添加 ANDROID_HOME 环境变量，也需要在这里添加。在“环境变量”窗口中，添加一个新的系统变量，名称为“ANDROID_HOME”，值为你的 Android SDK 路径。

然后以管理员身份重新启动命令行，这样你就可以在其中运行 react-native 命令。

3）创建你的项目 在命令行中，导航到你想放置项目的文件夹，运行以下命令：

```
react-native init ProjectName
```

```
react-native start
```

Section 1.3: Setup for Windows

Note: You cannot develop react-native apps for iOS on Windows, only react-native android apps.

The official setup docs for react-native on windows can be found here. If you need more details there is a granular guide here.

Tools/Environment

- Windows 10
- command line tool (eg Powershell or windows command line)
- Chocolatey (steps to setup via PowerShell)
- The JDK (version 8)
- Android Studio
- An Intel machine with Virtualization technology enabled for HAXM (optional, only needed if you want to use an emulator)

1) Setup your machine for react native development

Start the command line as an administrator run the following commands:

```
choco install nodejs.install
choco install python2
```

Restart command line as an administrator so you can run npm

```
npm install -g react-native-cli
```

After running the last command copy the directory that react-native was installed in. You will need this for Step 4. I tried this on two computers in one case it was: `C:\Program Files (x86)\Nodist\v-x64\6.2.2`. In the other it was: `C:\Users\admin\AppData\Roaming\npm`

2) Set your Environment Variables

A Step by Step guide with images can be found here for this section.

Open the Environment Variables window by navigating to:

[Right click] "Start" menu -> System -> Advanced System Settings -> Environment Variables

In the bottom section find the "Path" System Variable and add the location that react-native was installed to in step 1.

If you haven't added an ANDROID_HOME environment variable you will have to do that here too. While still in the "Environment Variables" window, add a new System Variable with the name "ANDROID_HOME" and value as the path to your android sdk.

Then restart the command line as an admin so you can run react-native commands in it.

3) Create your project In command line, navigate to the folder you want to place your project and run the following command:

```
react-native init ProjectName
```

4) 运行你的项目 从 Android Studio 启动一个模拟器，导航到项目根目录的命令行中并运行：

```
cd ProjectName
react-native run-android
```

你可能会遇到依赖问题。例如，可能会出现错误提示你没有正确的构建工具版本。为了解决这个问题，你需要打开 Android Studio 中的 SDK 管理器，并从那里下载构建工具。

恭喜！

要刷新界面，您可以在模拟器中运行应用时按两次 r 键。要查看开发者选项，您可以按 ctrl + m。

4) Run your project Start an emulator from android studio Navigate to the root directory of your project in command line and run it:

```
cd ProjectName
react-native run-android
```

You may run into dependency issues. For example, there may be an error that you do not have the correct build tools version. To fix this you will have to open the sdk manager in Android Studio and download the build tools from there.

Congrats!

To refresh the ui you can press the `r` key twice while in the emulator and running the app. To see developer options you can press `ctrl + m`.

第2章：你好，世界

第2.1节：编辑 index.ios.js 或 index.android.js

打开 `index.ios.js` 或 `index.android.js`，删除 **`<View> </View>`** 之间的所有内容。之后，写入 **`<Text>`** 你好，世界！ **`</Text>`** 并运行模拟器。

你应该能在屏幕上看到 你好，世界！

恭喜！你已经成功写出了第一个“你好，世界！”

第2.2节：你好，世界！

```
import React, { Component } from 'react';
import { AppRegistry, Text } from 'react-native';

class HelloWorldApp extends Component {
render() {
    return (
      <Text>你好，世界!</Text>
    );
  }
}

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

Chapter 2: Hello World

Section 2.1: Editing index.ios.js or index.android.js

Open `index.ios.js` or `index.android.js` and delete everything between the **`<View> </View>`**. After that, write **`<Text>`** `Hello World!` **`</Text>`** and run the emulator.

You should see `Hello World!` written on the screen!

Congrats! You've successfully written your first Hello World!

Section 2.2: Hello world!

```
import React, { Component } from 'react';
import { AppRegistry, Text } from 'react-native';

class HelloWorldApp extends Component {
  render() {
    return (
      <Text>Hello world!</Text>
    );
  }
}

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

第3章：Props

Props，或称属性，是传递给React应用中子组件的数据。React组件根据它们的props和内部状态渲染UI元素。组件所接受（并使用）的props定义了它如何被外部控制。

第3.1节：PropTypes

prop-types 包允许你为组件添加运行时类型检查，以确保传递给组件的props类型正确。例如，如果你没有向下面的组件传递 name 或 isYummy prop，在开发模式下会抛出错误。在生产模式下不会进行prop类型检查。定义 propTypes 可以使你的组件更易读且易于维护。

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { AppRegistry, Text, View } from 'react-native';

import styles from './styles.js';

class Recipe extends Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    isYummy: PropTypes.bool.isRequired
  }
render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.name}</Text>
        {this.props.isYummy ? <Text>这个食谱很好吃</Text> : null}
      </View>
    )
  }
}

AppRegistry.registerComponent('Recipe', () => Recipe);


// 使用该组件
<Recipe name="煎饼" isYummy={true} />
```

多个 PropTypes

你也可以为一个 props 设置多个 propTypes。例如，我接收的 name props 也可以是一个对象，我可以这样写。

```
static propTypes = {
  name: PropTypes.oneOfType([
      PropTypes.string,
PropTypes.object
  ])
}
```

子组件属性

还有一个特殊的属性叫做children，它不是像这样传入的

Chapter 3: Props

Props, or properties, are data that is passed to child components in a React application. React components render UI elements based on their props and their internal state. The props that a component takes (and uses) defines how it can be controlled from the outside.

Section 3.1: PropTypes

The prop-types package allows you to add runtime type checking to your component that ensures the types of the props passed to the component are correct. For instance, if you don't pass a name or isYummy prop to the component below it will throw an error in development mode. In production mode the prop type checks are not done. Defining propTypes can make your component more readable and maintainable.

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { AppRegistry, Text, View } from 'react-native';

import styles from './styles.js';

class Recipe extends Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    isYummy: PropTypes.bool.isRequired
  }
  render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.name}</Text>
        {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
      </View>
    )
  }
}

AppRegistry.registerComponent('Recipe', () => Recipe);


// Using the component
<Recipe name="Pancakes" isYummy={true} />
```

Multiple PropTypes

You can also have multiple propTypes for one props. For example, the name props I'm taking can also be an object, I can write it as.

```
static propTypes = {
  name: PropTypes.oneOfType([
      PropTypes.string,
      PropTypes.object
  ])
}
```

Children Props

There is also a special props called children, which is **not** passed in like

```
<Recipe children={something}/>
```

相反，你应该这样做

```
<Recipe>
  <Text>Hello React Native</Text>
</Recipe>
```

然后你可以在 Recipe 的 render 中这样写：

```
return (
  <View style={styles.container}>
    {this.props.children}
    {this.props.isYummy ? <Text>这个食谱很好吃</Text> : null}
  </View>
)
```

你会在你的Recipe中看到一个<Text>组件，显示Hello React Native，挺酷的吧？

子组件的propType是

```
children: PropTypes.node
```

第3.2节：什么是props？

Props用于将数据从父组件传递到子组件。Props是只读的。子组件只能使用this.props.keyName获取父组件传递的props。通过props，可以使组件可复用。

第3.3节：props的使用

设置完成后，将以下代码复制到index.android.js或index.ios.js文件中以使用props。

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <问候 name='Jaina' />
        <问候 name='Valeera' />
      </视图>
    );
  }
}

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

```
<Recipe children={something}/>
```

Instead, you should do this

```
<Recipe>
  <Text>Hello React Native</Text>
</Recipe>
```

then you can do this in Recipe's render:

```
return (
  <View style={styles.container}>
    {this.props.children}
    {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
  </View>
)
```

You will have a **<Text>** component in your `Recipe` saying `Hello React Native`, pretty cool hum?

And the propType of children is

```
children: PropTypes.node
```

Section 3.2: What are props?

Props are used to transfer data from parent to child component. Props are read only. Child component can only get the props passed from parent using **this.props.keyName**. Using props one can make his component reusable.

Section 3.3: Use of props

Once setup is completed. Copy the code below to `index.android.js` or to `index.ios.js` file to use the props.

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}

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

使用props可以使组件通用。例如，你有一个按钮组件。你可以向该组件传递不同的props，这样就可以将该按钮放置在视图中的任何位置。

来源：Props-React Native

第3.4节：默认Props

defaultProps允许你为组件设置默认的prop值。在下面的例子中，如果你没有传递name prop，它将显示John，否则显示传入的值

```
class Example extends Component {
  render() {
    return (
      <View>
        <Text>{this.props.name}</Text>
      </View>
    )
  }
}


示例。defaultProps = {
  name: 'John'
}
```

Using props one can make his component generic. For example, you have a Button component. You can pass different props to that component, so that one can place that button anywhere in his view.

source: Props-React Native

Section 3.4: Default Props

defaultProps allows you to set default prop values for your component. In the below example if you do not pass the name props, it will display John otherwise it will display the passed value

```
class Example extends Component {
  render() {
    return (
      <View>
        <Text>{this.props.name}</Text>
      </View>
    )
  }
}


Example.defaultProps = {
  name: 'John'
}
```

第4章：多属性渲染

第4.1节：渲染多个变量

对于渲染多个属性或变量，我们可以使用``。

```
render() {
    let firstName = 'test';
    let lastName = 'name';
    return (
      <View style={styles.container}>
        <Text>{`${firstName} ${lastName}` } </Text>
      </View>
    );
  }
```

输出：test name

Chapter 4: Multiple props rendering

Section 4.1: render multiple variables

For rendering multiple props or variables we can use ``.

```
render() {
    let firstName = 'test';
    let lastName = 'name';
    return (
      <View style={styles.container}>
        <Text>{`${firstName} ${lastName}` } </Text>
      </View>
    );
  }
```

Output: test name

第5章：模态框

属性	详细信息
animationType	它是一个枚举，包含（'none'，'slide'，'fade'），用于控制模态动画。
visible	它是一个布尔值，用于控制模态的可见性。
onShow	允许传入一个函数，该函数将在模态显示后被调用。
transparent	用于设置透明度的布尔值。
onRequestClose (android)	始终定义一个方法，当用户点击返回按钮时调用该方法。
onOrientationChange（**IOS**）	总是定义一个在方向变化时调用的方法
supportedOrientations（**IOS**）	枚举('portrait'，'portrait-upside-down'，'landscape'，'landscape-left'，'landscape-right')

Modal 组件是一种在包裹视图之上展示内容的简单方式。

第5.1节：Modal 基本示例

```
import React, { Component } from 'react';
import {
Modal,
Text,
View,
Button,
StyleSheet,
} from 'react-native';

const styles = StyleSheet.create({
  mainContainer: {
marginTop: 22,
  },
modalContainer: {
    marginTop: 22,
  },
});

class Example extends Component {
  constructor() {
    super();
    this.state = {
      visibility: false,
    };
  }


setModalVisibility(visible) {
    this.setState({
      visibility: visible,
    });
  }

render() {
    return (
      <View style={styles.mainContainer}>
        <Modal
animationType={'slide'}
          transparent={false}
          visible={this.state.visibility}
        >
          <View style={styles.modalContainer}>
            <View>
```

Chapter 5: Modal

Prop	details
animationType	it's an enum of ('**none**', '**slide**', '**fade**') and it controls modal animation.
visible	its a bool that controls modal visiblity.
onShow	it allows passing a function that will be called once the modal has been shown.
transparent	bool to set transparency.
onRequestClose (**android**)	it always defining a method that will be called when user tabs back button
onOrientationChange (**IOS**)	it always defining a method that will be called when orientation changes
supportedOrientations (**IOS**)	enum('portrait', 'portrait-upside-down', 'landscape', 'landscape-left', 'landscape-right')

Modal component is a simple way to present content above an enclosing view.

Section 5.1: Modal Basic Example

```
import React, { Component } from 'react';
import {
  Modal,
  Text,
  View,
  Button,
  StyleSheet,
} from 'react-native';

const styles = StyleSheet.create({
  mainContainer: {
    marginTop: 22,
  },
  modalContainer: {
    marginTop: 22,
  },
});

class Example extends Component {
  constructor() {
    super();
    this.state = {
      visibility: false,
    };
  }


  setModalVisibility(visible) {
    this.setState({
      visibility: visible,
    });
  }

  render() {
    return (
      <View style={styles.mainContainer}>
        <Modal
          animationType={'slide'}
          transparent={false}
          visible={this.state.visibility}
        >
          <View style={styles.modalContainer}>
            <View>
```

```
              <Text>I'm a simple Modal</Text>
              <Button
                color="#000"
onPress={() => this.setModalVisibility(!this.state.visibility)}
              title="隐藏模态框"
              />
            </View>
          </View>
        </Modal>

        <Button
          color="#000"
onPress={() => this.setModalVisibility(true)}
          title="显示模态框"
/>
      </View>
    );
  }
}

export default Example;
```

第5.2节：透明模态示例

请参见此示例 here。

```
import React, { Component } from 'react';
import { Text, View, StyleSheet, Button, Modal } from 'react-native';
import { Constants } from 'expo';

export default class App extends Component {
  state = {
modalVisible: false,
  };

_handleButtonPress = () => {
    this.setModalVisible(true);
  };

setModalVisible = (visible) => {
    this.setState({modalVisible: visible});
  }

render() {
    var modalBackgroundStyle = {
      backgroundColor: 'rgba(0, 0, 0, 0.5)'
    };
    var innerContainerTransparentStyle = {backgroundColor: '#fff', padding: 20};
    return (
      <View style={styles.container}>
      <Modal
animationType='fade'
          transparent={true}
visible={this.state.modalVisible}
          onRequestClose={() => this.setModalVisible(false)}
          >
          <View style={[styles.container, modalBackgroundStyle]}>
            <View style={innerContainerTransparentStyle}>
              <Text>这是一个模态框</Text>
              <Button title='关闭'
onPress={this.setModalVisible.bind(this, false)}/>
```

```
              <Text>I'm a simple Modal</Text>
              <Button
                color="#000"
                onPress={() => this.setModalVisibility(!this.state.visibility)}
                title="Hide Modal"
              />
            </View>
          </View>
        </Modal>

        <Button
          color="#000"
          onPress={() => this.setModalVisibility(true)}
          title="Show Modal"
        />
      </View>
    );
  }
}

export default Example;
```

Section 5.2: Transparent Modal Example

See this example here.

```
import React, { Component } from 'react';
import { Text, View, StyleSheet, Button, Modal } from 'react-native';
import { Constants } from 'expo';

export default class App extends Component {
  state = {
    modalVisible: false,
  };

  _handleButtonPress = () => {
    this.setModalVisible(true);
  };

  setModalVisible = (visible) => {
    this.setState({modalVisible: visible});
  }

  render() {
    var modalBackgroundStyle = {
      backgroundColor: 'rgba(0, 0, 0, 0.5)'
    };
    var innerContainerTransparentStyle = {backgroundColor: '#fff', padding: 20};
    return (
      <View style={styles.container}>
      <Modal
          animationType='fade'
          transparent={true}
          visible={this.state.modalVisible}
          onRequestClose={() => this.setModalVisible(false)}
          >
          <View style={[styles.container, modalBackgroundStyle]}>
            <View style={innerContainerTransparentStyle}>
              <Text>This is a modal</Text>
              <Button title='close'
                onPress={this.setModalVisible.bind(this, false)}/>
            <视图>
          <视图>
        <Modal>
        <Button
title="按我"
          onPress={this._handleButtonPress}
        />

      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
flex: 1,
alignItems: 'center',
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#ecf0f1',
  }
});
```

```
            </View>
          </View>
        </Modal>
        <Button
          title="Press me"
          onPress={this._handleButtonPress}
        />

      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#ecf0f1',
  }
});
```

第6章：状态

第6.1节：setState

要更改应用程序中的视图，可以使用setState——这将重新渲染你的组件及其所有子组件。setState 会在新状态和之前状态之间执行浅合并，并触发组件的重新渲染。

setState 接受一个键值对象或返回键值对象的函数

键值对象

```
this.setState({myKey: 'myValue'});
```

函数

使用函数对于基于现有状态或属性更新值非常有用。

```
this.setState((previousState, currentProps) => {
    return {
myInteger: previousState.myInteger+1
    }
})
```

你也可以传入一个可选的回调函数给setState，当组件使用新状态重新渲染后该回调函数将被触发。

```
this.setState({myKey: 'myValue'}, () => {
    // 组件已重新渲染... 做一些了不起的事情！
});
```

完整示例

```
import React, { Component } from 'react';
import { AppRegistry, StyleSheet, Text, View, TouchableOpacity } from 'react-native';

export default class MyParentComponent extends Component {
constructor(props) {
    super(props);

    this.state = {
      myInteger: 0
    }

  }
getRandomInteger() {
    const randomInt = Math.floor(Math.random()*100);

    this.setState({
myInteger: randomInt
    });

  }
incrementInteger() {

    this.setState((previousState, currentProps) => {
      return {
myInteger: previousState.myInteger+1
      }
```

Chapter 6: State

Section 6.1: setState

To change view in your application you can use `setState` - this will re-render your component and any of its child components. setState performs a shallow merge between the new and previous state, and triggers a re-render of the component.

`setState` takes either a key-value object or a function that returns a key-value object

Key-Value Object

```
this.setState({myKey: 'myValue'});
```

Function

Using a function is useful for updating a value based off the existing state or props.

```
this.setState((previousState, currentProps) => {
    return {
        myInteger: previousState.myInteger+1
    }
})
```

You can also pass an optional callback to `setState` that will be fired when the component has re-rendered with the new state.

```
this.setState({myKey: 'myValue'}, () => {
    // Component has re-rendered... do something amazing!
});
```

Full Example

```
import React, { Component } from 'react';
import { AppRegistry, StyleSheet, Text, View, TouchableOpacity } from 'react-native';

export default class MyParentComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      myInteger: 0
    }

  }
  getRandomInteger() {
    const randomInt = Math.floor(Math.random()*100);

    this.setState({
      myInteger: randomInt
    });

  }
  incrementInteger() {

    this.setState((previousState, currentProps) => {
      return {
        myInteger: previousState.myInteger+1
      }
    });

  }
render() {

    return <View style={styles.container}>

      <Text>父组件整数: {this.state.myInteger}</Text>

      <MyChildComponent myInteger={this.state.myInteger} />

      <Button label="获取随机整数" onPress={this.getRandomInteger.bind(this)} />
      <Button label="递增整数" onPress={this.incrementInteger.bind(this)} />

    <视图>

  }
}

export default class MyChildComponent extends Component {
  constructor(props) {
    super(props);
  }
render() {

    // 当“MyParentComponent”状态变化时，这里会更新
    返回 <视图>
      <Text>子组件整数: {this.props.myInteger}</Text>
    </View>

  }
}

export default class Button extends Component {
  constructor(props) {
    super(props);
  }
render() {

    return <TouchableOpacity onPress={this.props.onPress}>
        <View style={styles.button}>
          <Text style={styles.buttonText}>{this.props.label}</Text>
        </View>
      </TouchableOpacity>

  }
}

const styles = StyleSheet.create({
  container: {
flex: 1,
justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
button: {
backgroundColor: '#444',
    padding: 10,
marginTop: 10
  },
buttonText: {
    color: '#fff'
```

```
    });

  }
  render() {

    return <View style={styles.container}>

      <Text>Parent Component Integer: {this.state.myInteger}</Text>

      <MyChildComponent myInteger={this.state.myInteger} />

      <Button label="Get Random Integer" onPress={this.getRandomInteger.bind(this)} />
      <Button label="Increment Integer" onPress={this.incrementInteger.bind(this)} />

    </View>

  }
}

export default class MyChildComponent extends Component {
  constructor(props) {
    super(props);
  }
  render() {

    // this will get updated when "MyParentComponent" state changes
    return <View>
      <Text>Child Component Integer: {this.props.myInteger}</Text>
    </View>

  }
}

export default class Button extends Component {
  constructor(props) {
    super(props);
  }
  render() {

    return <TouchableOpacity onPress={this.props.onPress}>
        <View style={styles.button}>
          <Text style={styles.buttonText}>{this.props.label}</Text>
        </View>
      </TouchableOpacity>

  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  button: {
    backgroundColor: '#444',
    padding: 10,
    marginTop: 10
  },
  buttonText: {
    color: '#fff'
  }
});

AppRegistry.registerComponent('MyApp', () => MyParentComponent);
```

第6.2节：初始化状态

你应该在组件的构造函数中这样初始化状态：

```
export default class MyComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      myInteger: 0
    }
  }
render() {
    return  (
      <View>
        <Text>整数: {this.state.myInteger}</Text>
      </View>
    )
  }
}
```

使用 setState 可以更新视图。

```
  }
});

AppRegistry.registerComponent('MyApp', () => MyParentComponent);
```

Section 6.2: Initialize State

You should initialize state inside the constructor function of your component like this:

```
export default class MyComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      myInteger: 0
    }
  }
  render() {
    return  (
      <View>
        <Text>Integer: {this.state.myInteger}</Text>
      </View>
    )
  }
}
```

Using setState one can update the view.

第7章：路由

路由或导航允许应用在不同屏幕之间切换。它对移动应用至关重要，因为它为用户提供了他们所在位置的上下文，解耦了屏幕之间的用户操作并实现屏幕间的切换，提供了整个应用的状态机模型。

第7.1节：导航组件

Navigator 适用于 IOS 和安卓。

```
import React, { Component } from 'react';
import { Text, Navigator, TouchableHighlight } from 'react-native';

export default class NavAllDay extends Component {
render() {
    return (
      <Navigator
initialRoute={{ title: 'Awesome Scene', index: 0 }}
        renderScene={(route, navigator) =>
          <Text>Hello {route.title}!</Text>
        }
style={{padding: 100}}
      />
    );
  }
}
```

传递给Navigator的路由是以对象形式提供的。你还需要提供一个renderScene函数，用于渲染每个路由对象对应的场景。 initialRoute用于指定第一个路由。

Chapter 7: Routing

Routing or navigation allows applications to between different screens. Its vital to a mobile app as it provides context to user about where they are, decouple user actions between screens and move between them, provide a state machine like model of the whole app.

Section 7.1: Navigator component

Navigator works for both IOS and android.

```
import React, { Component } from 'react';
import { Text, Navigator, TouchableHighlight } from 'react-native';

export default class NavAllDay extends Component {
  render() {
    return (
      <Navigator
        initialRoute={{ title: 'Awesome Scene', index: 0 }}
        renderScene={(route, navigator) =>
          <Text>Hello {route.title}!</Text>
        }
        style={{padding: 100}}
      />
    );
  }
}
```

Routes to `Navigator` are provided as objects. You also provide a `renderScene` function that renders the scene for each route object. `initialRoute` is used to specify the first route.

第8章：样式

样式在一个 JSON 对象中定义，属性名称类似于 CSS 中的样式属性。这样的对象可以直接内联写在组件的 style 属性中，或者传递给函数StyleSheet.create(StyleObject)并存储在变量中，通过类似 CSS 类的选择器名称来简化内联访问。

第8.1节：条件样式

```
<View style={[(this.props.isTrue) ? styles.bgcolorBlack : styles.bgColorWhite]}>
```

如果 isTrue 的值为 true，则背景颜色为黑色，否则为白色。

第8.2节：使用内联样式进行样式设置

每个React Native组件都可以接受一个 style 属性。你可以传入一个包含CSS风格样式属性的JavaScript对象：

```
<Text style={{color:'red'}}>红色文本</Text>
```

这可能效率较低，因为每次组件渲染时都要重新创建该对象。推荐使用样式表。

第8.3节：使用样式表进行样式设置

```
import React, { Component } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    red: {
color: 'red'
    },
big: {
fontSize: 30
    }
});

class 示例 extends 组件 {
   render() {
        return (
            <View>
                <Text style={styles.red}>红色</Text>
                <Text style={styles.big}>大号</Text>
            </View>
        );
    }
}
```

StyleSheet.create() 返回一个对象，其中的值是数字。React Native 知道将这些数字 ID 转换为正确的样式对象。

第8.4节：添加多重样式

你可以传递一个数组给 style 属性以应用多重样式。当存在冲突时，列表中最后一个样式优先。

Chapter 8: Styling

Styles are defined within a JSON object with similar styling attribute names like in CSS. Such an object can either be put inline in the style prop of a component or it can be passed to the function `StyleSheet.create(StyleObject)` and be stored in a variable for shorter inline access by using a selector name for it similar to a class in CSS.

Section 8.1: Conditional Styling

```
<View style={[(this.props.isTrue) ? styles.bgcolorBlack : styles.bgColorWhite]}>
```

If the value of `isTrue` is **`true`** then it will have black background color otherwise white.

Section 8.2: Styling using inline styles

Each React Native component can take a `style` prop. You can pass it a JavaScript object with CSS-style style properties:

```
<Text style={{color:'red'}}>Red text</Text>
```

This can be inefficient as it has to recreate the object each time the component is rendered. Using a stylesheet is preferred.

Section 8.3: Styling using a stylesheet

```
import React, { Component } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    red: {
        color: 'red'
    },
    big: {
        fontSize: 30
    }
});

class Example extends Component {
    render() {
        return (
            <View>
                <Text style={styles.red}>Red</Text>
                <Text style={styles.big}>Big</Text>
            </View>
        );
    }
}
```

`StyleSheet.create()` returns an object where the values are numbers. React Native knows to convert these numeric IDs into the correct style object.

Section 8.4: Adding multiple styles

You can pass an array to the `style` prop to apply multiple styles. When there is a conflict, the last one in the list takes precedence.

```
import React, { Component } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    red: {
color: 'red'
    },
greenUnderline: {
        color: 'green',
        textDecoration: 'underline'
    },
big: {
fontSize: 30
    }
});

class 示例 extends 组件 {
   render() {
        return (
            <View>
                <Text style={[styles.red, styles.big]}>大号红色</Text>
                <Text style={[styles.red, styles.greenUnderline]}>绿色下划线</Text>
                <Text style={[styles.greenUnderline, styles.red]}>红色下划线</Text>
                <Text style={[styles.greenUnderline, styles.red, styles.big]}>大号红色
下划线</Text>
                <Text style={[styles.big, {color:'yellow'}]}>大号黄色</Text>
            </View>
        );
    }
}
```

```
import React, { Component } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    red: {
        color: 'red'
    },
    greenUnderline: {
        color: 'green',
        textDecoration: 'underline'
    },
    big: {
        fontSize: 30
    }
});

class Example extends Component {
    render() {
        return (
            <View>
                <Text style={[styles.red, styles.big]}>Big red</Text>
                <Text style={[styles.red, styles.greenUnderline]}>Green underline</Text>
                <Text style={[styles.greenUnderline, styles.red]}>Red underline</Text>
                <Text style={[styles.greenUnderline, styles.red, styles.big]}>Big red
underline</Text>
                <Text style={[styles.big, {color:'yellow'}]}>Big yellow</Text>
            </View>
        );
    }
}
```

第9章：布局

第9.1节：弹性盒子（Flexbox）

弹性盒子是一种布局模式，用于安排页面上的元素，使得当页面布局需要适应不同屏幕尺寸和不同显示设备时，元素的行为是可预测的。默认情况下，弹性盒子将子元素排列成一列。但你可以使用flexDirection: 'row'将其改为行排列。

flexDirection

```
const Direction = (props)=>{
  return (
    <View style={styles.container}>
      <Box/>
      <Box/>
      <Box/>
      <View style={{flexDirection:'row'}}>
        <Box/>
        <Box/>
        <Box/>
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
flex:1,
backgroundColor: '#AED581',
  }
});
```

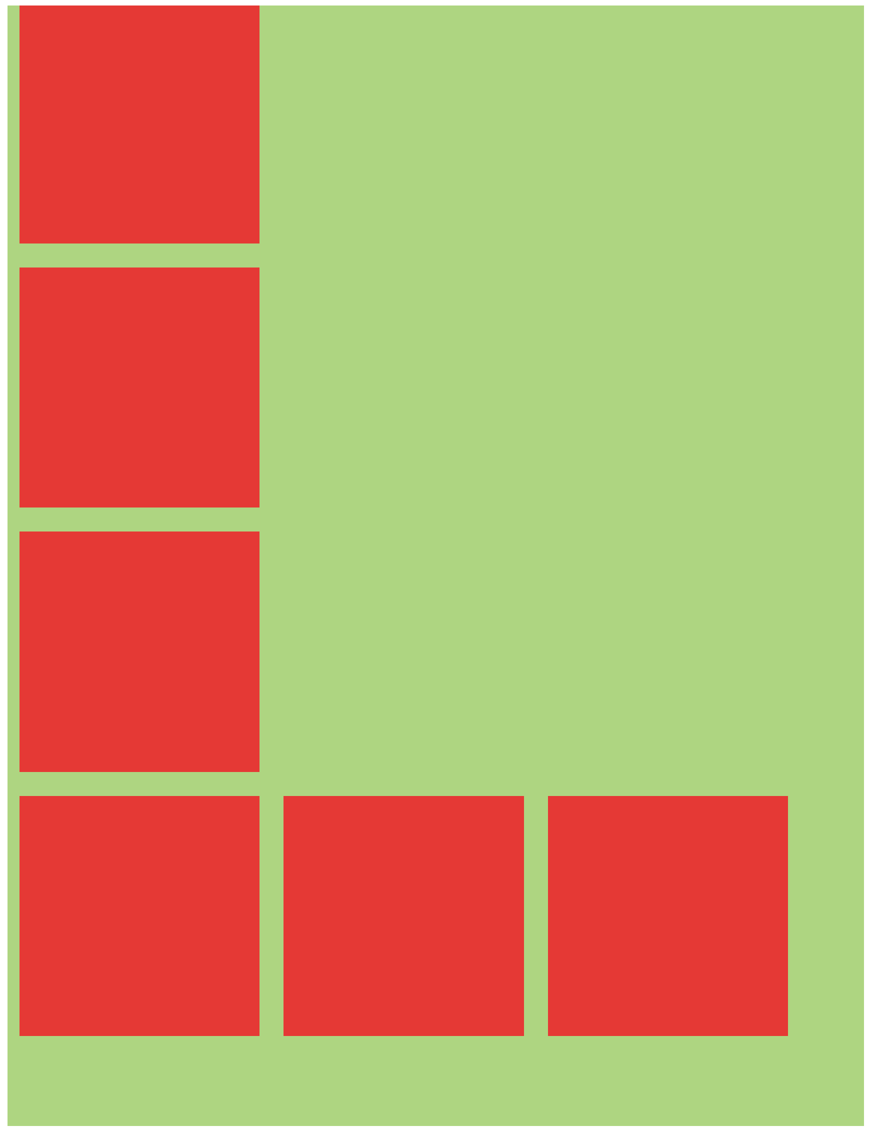
Chapter 9: Layout

Section 9.1: Flexbox

Flexbox is a layout mode providing for the arrangement of elements on a page such that the elements behave predictably when the page layout must accommodate different screen sizes and different display devices. By default flexbox arranges children in a column. But you can change it to row using `flexDirection: 'row'`.

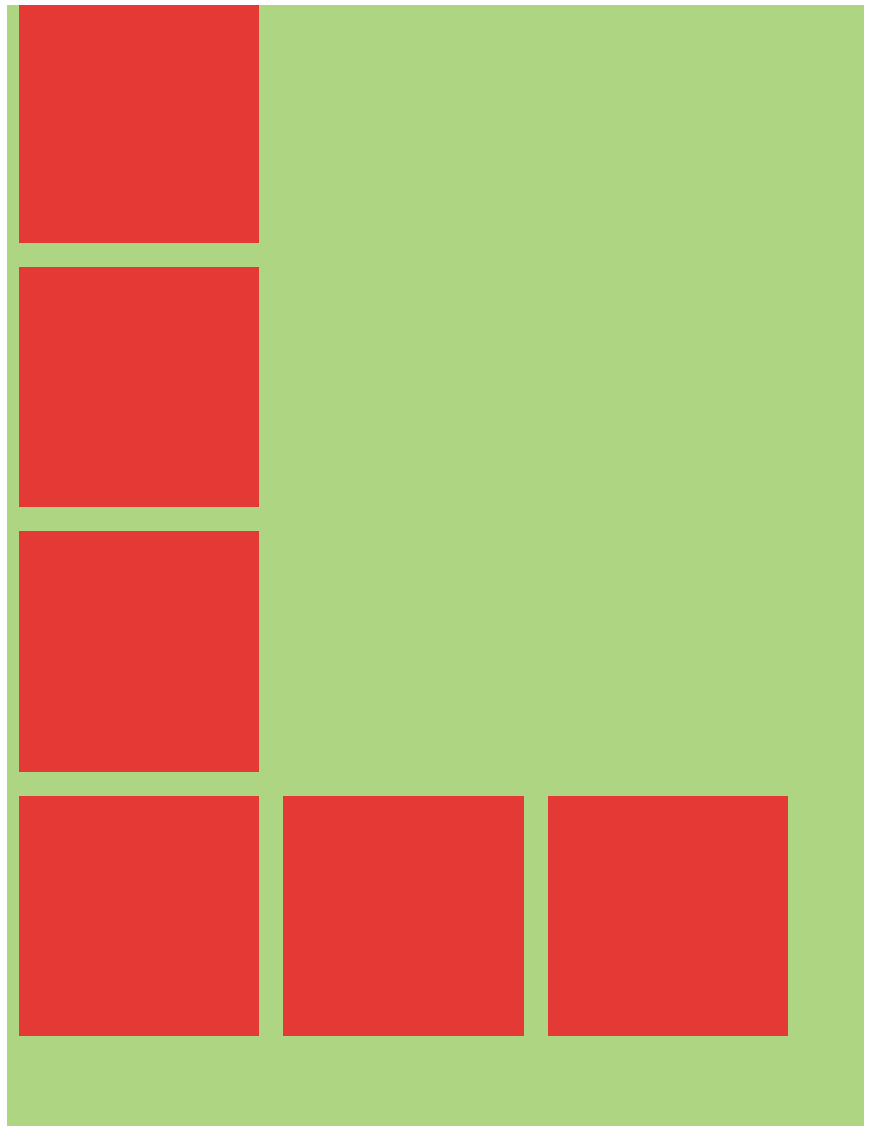
flexDirection

```
const Direction = (props)=>{
  return (
    <View style={styles.container}>
      <Box/>
      <Box/>
      <Box/>
      <View style={{flexDirection:'row'}}>
        <Box/>
        <Box/>
        <Box/>
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: '#AED581',
  }
});
```



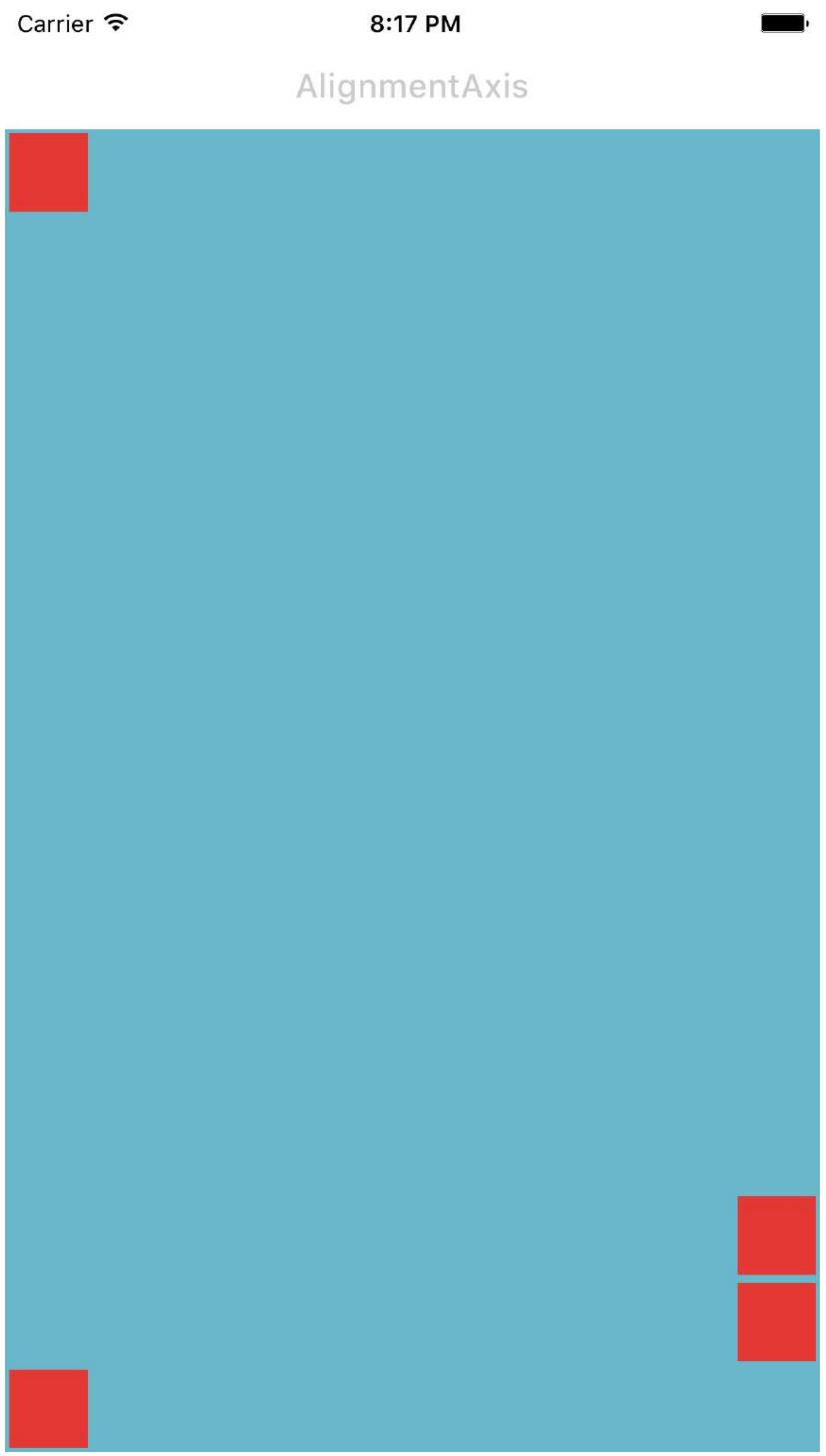
对齐轴

```
const AlignmentAxis = (props)=>{
  return (
    <View style={styles.container}>
      <Box />
      <View style={{flex:1, alignItems:'flex-end', justifyContent:'flex-end'}}>
        <Box />
        <Box />
      </View>
      <Box />
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
flex:1,
backgroundColor: `#69B8CC`,
  },
text:{
color: 'white',
    textAlign:'center'
  }
});
```



Alignment axis

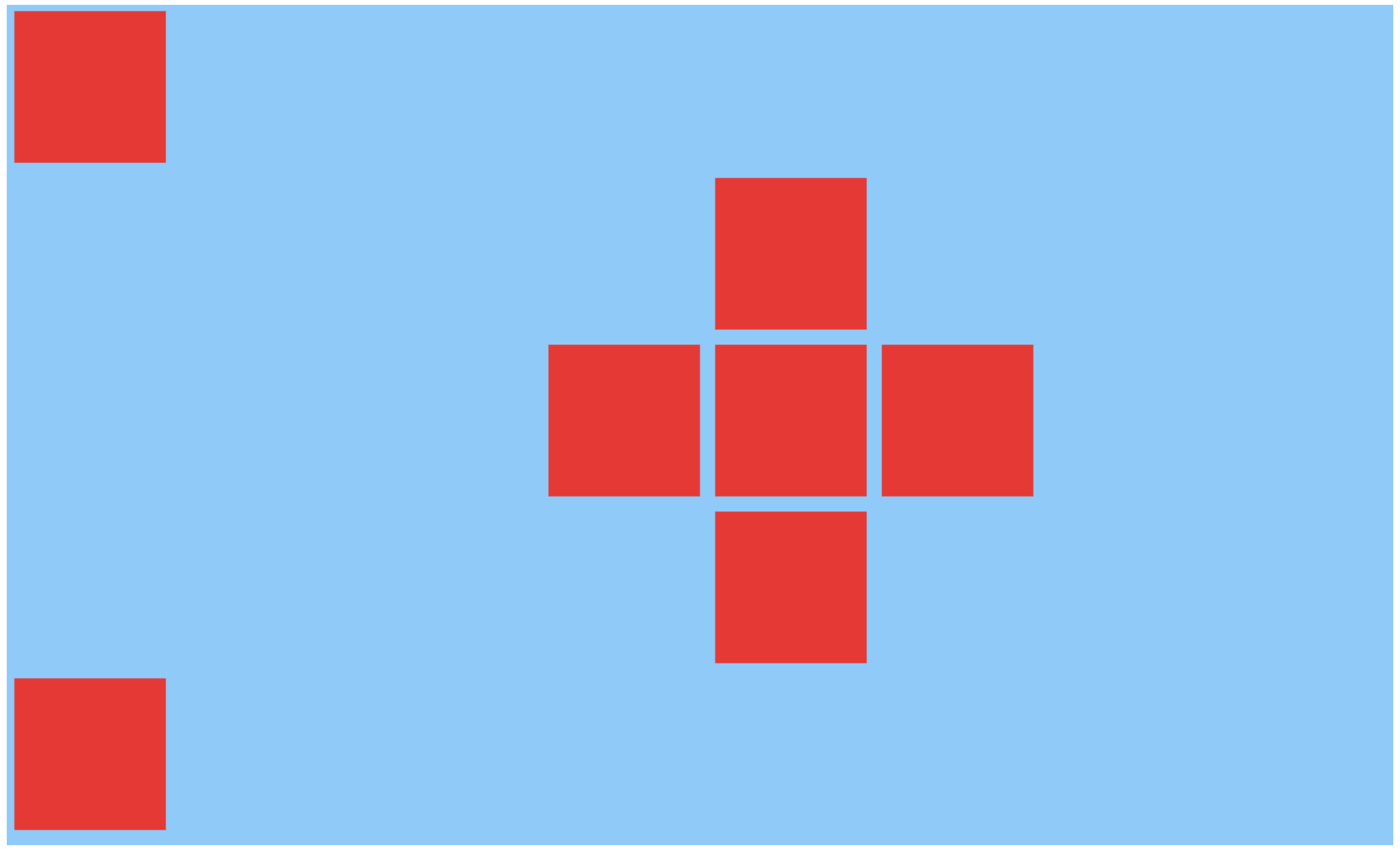
```
const AlignmentAxis = (props)=>{
  return (
    <View style={styles.container}>
      <Box />
      <View style={{flex:1, alignItems:'flex-end', justifyContent:'flex-end'}}>
        <Box />
        <Box />
      </View>
      <Box />
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: `#69B8CC`,
  },
  text:{
    color: 'white',
    textAlign:'center'
  }
});
```


Carrier
8:17 PM
AlignmentAxis

Carrier
8:17 PM
AlignmentAxis

对齐

```
const Alignment = (props)=>{
  return (
    <View style={styles.container}>
      <Box/>
      <View style={{alignItems:'center'}}>
        <Box/>
        <View style={{flexDirection:'row'}}>
          <Box/>
          <Box/>
          <Box/>
        </View>
        <Box/>
      </View>
      <Box/>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
flex:1,
backgroundColor: `#69B8CC`,
  },
text:{
color: 'white',
    textAlign:'center'
  }
});
```

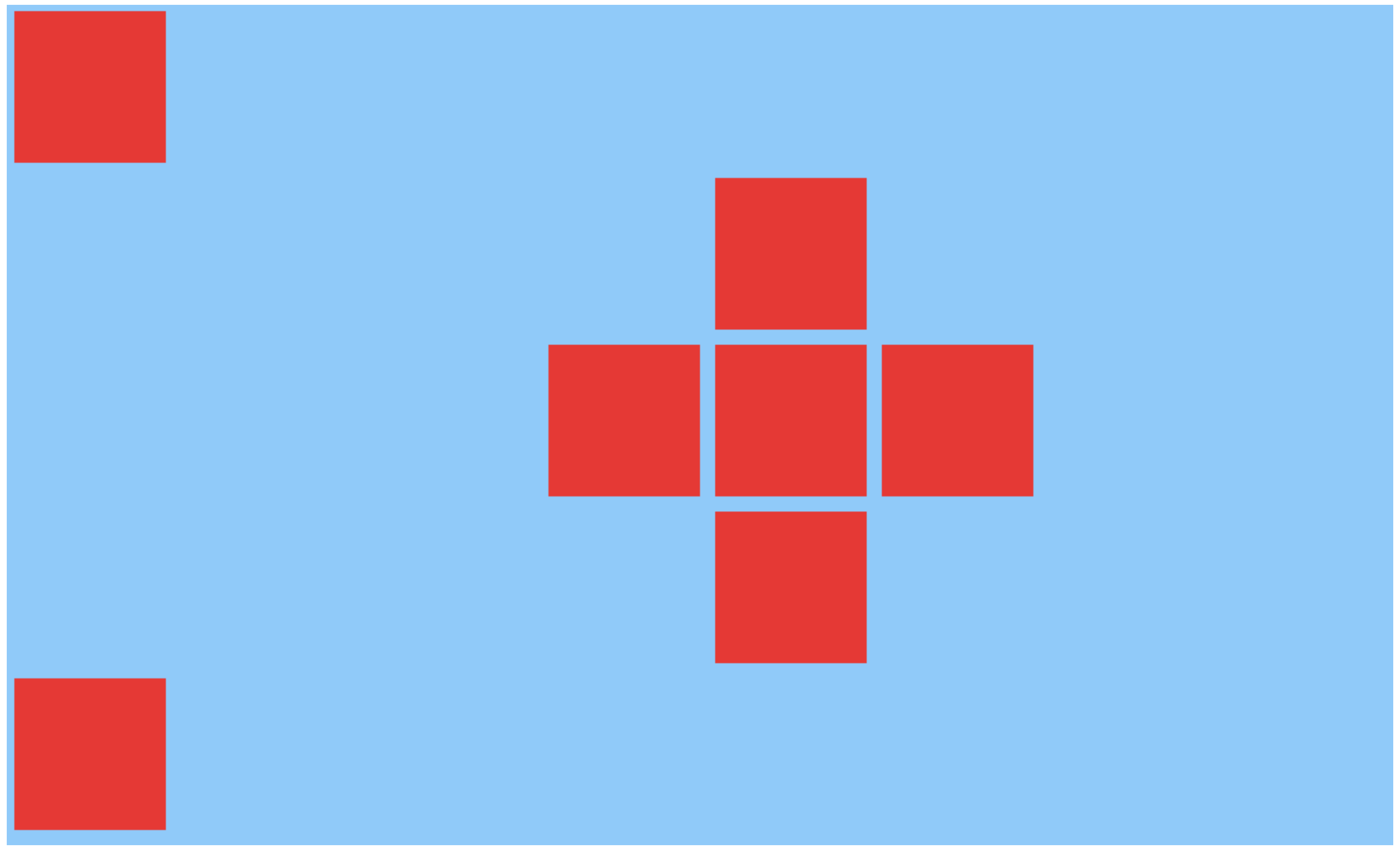


弹性尺寸

```
const FlexSize = (props)=>{
  return (
```

Alignment

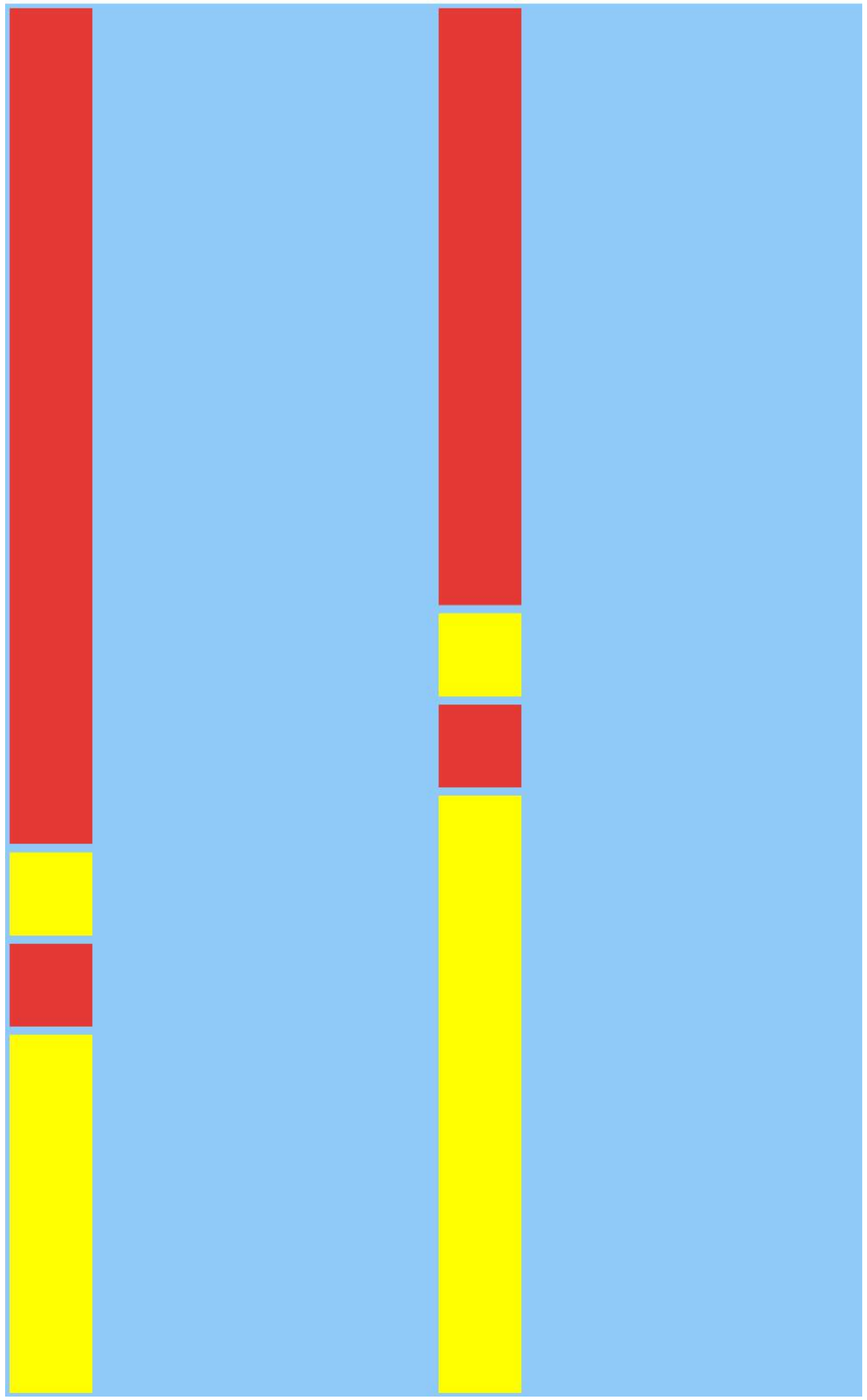
```
const Alignment = (props)=>{
  return (
    <View style={styles.container}>
      <Box/>
      <View style={{alignItems:'center'}}>
        <Box/>
        <View style={{flexDirection:'row'}}>
          <Box/>
          <Box/>
          <Box/>
        </View>
        <Box/>
      </View>
      <Box/>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: `#69B8CC`,
  },
  text:{
    color: 'white',
    textAlign:'center'
  }
});
```

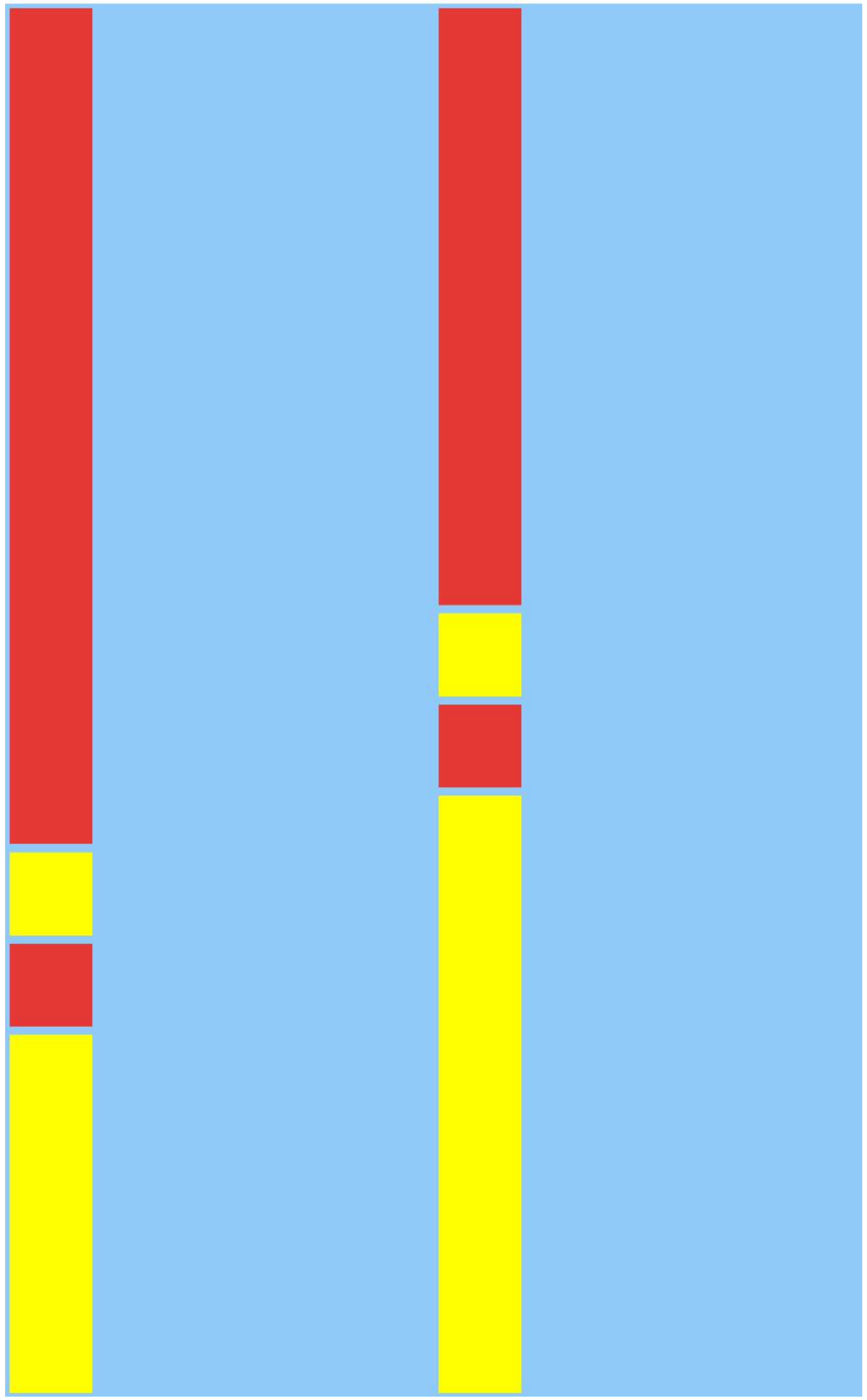


Flex size

```
const FlexSize = (props)=>{
  return (
    <View style={styles.container}>
      <View style={{flex:0.1}}>
        <Box style={{flex:0.7}}/>
        <Box style={{backgroundColor: 'yellow'}}/>
        <Box/>
        <Box style={{flex:0.3, backgroundColor: 'yellow'}}/>
      </View>
      <View style={{flex:0.1}}>
        <Box style={{flex:1}}/>
        <Box style={{backgroundColor: 'yellow'}}/>
        <Box/>
        <Box style={{flex:1, backgroundColor: 'yellow'}}/>
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
flex:1,
flexDirection:'row',
    backgroundColor: colors[1],
  },
});
```

```
    <View style={styles.container}>
      <View style={{flex:0.1}}>
        <Box style={{flex:0.7}}/>
        <Box style={{backgroundColor: 'yellow'}}/>
        <Box/>
        <Box style={{flex:0.3, backgroundColor: 'yellow'}}/>
      </View>
      <View style={{flex:0.1}}>
        <Box style={{flex:1}}/>
        <Box style={{backgroundColor: 'yellow'}}/>
        <Box/>
        <Box style={{flex:1, backgroundColor: 'yellow'}}/>
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    flexDirection:'row',
    backgroundColor: colors[1],
  },
});
```





关于 Facebook 的 flexbox 实现的更多信息 here。

More about Facebook's flexbox implementation here.

第10章：组件

第10.1节：基础组件

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'

class Example extends Component {
render () {
    return (
      <View>
        <Text> 我是一个基础组件 </Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

第10.2节：有状态组件

这些组件将具有变化的状态。

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'

class Example extends Component {
constructor (props) {
    super(props)
    this.state = {
      name: "Sriraman"
    }
  }
render () {
    return (
      <View>
        <Text> 你好, {this.state.name}</Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

第10.3节：无状态组件

顾名思义，无状态组件没有任何本地状态。它们也被称为哑组件
组件。由于没有本地状态，这些组件不需要生命周期方法，也不需要带有状态组件的大量样板代码。

不需要使用类语法，你可以简单地写成const name = ({props}) => (...)。通常无状态组件因此更加简洁。

下面是两个无状态组件App和Title的示例，并演示了组件之间传递props的方式：

Chapter 10: Components

Section 10.1: Basic Component

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'

class Example extends Component {
  render () {
    return (
      <View>
        <Text> I'm a basic Component </Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

Section 10.2: Stateful Component

These components will have changing States.

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'

class Example extends Component {
  constructor (props) {
    super(props)
    this.state = {
      name: "Sriraman"
    }
  }
  render () {
    return (
      <View>
        <Text> Hi, {this.state.name}</Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

Section 10.3: Stateless Component

As the name implies, Stateless Components do not have any local state. They are also known as **Dumb Components**. Without any local state, these components do not need lifecycle methods or much of the boilerplate that comes with a stateful component.

Class syntax is not required, you can simply do `const name = ({props}) => ( ... )`. Generally stateless components are more concise as a result.

Beneath is an example of two stateless components `App` and `Title`, with a demonstration of passing props between components:

```
import React from 'react'
import { View, Text, AppRegistry } from 'react-native'

const Title = ({Message}) => (
  <Text>{Message}</Text>
)

const App = () => (
  <View>
    <Title title='示例无状态组件' />
  </View>
)

AppRegistry.registerComponent('App', () => App)
```

这是推荐的组件模式（如果可能的话）。因为未来可以对这些组件进行优化，减少内存分配和不必要的检查。

```
import React from 'react'
import { View, Text, AppRegistry } from 'react-native'

const Title = ({Message}) => (
  <Text>{Message}</Text>
)

const App = () => (
  <View>
    <Title title='Example Stateless Component' />
  </View>
)

AppRegistry.registerComponent('App', () => App)
```

This is the recommended pattern for components, when possible. As in the future optimisations can be made for these components, reducing memory allocations and unnecessary checks.

第11章：ListView

第11.1节：简单示例

ListView - 一个核心组件，设计用于高效显示垂直滚动的动态数据列表。最简API是创建一个ListView.DataSource，用一个简单的数据块数组填充它，然后实例化一个ListView组件，传入该数据源和一个renderRow回调函数，该函数从数据数组中取出一个数据块并返回一个可渲染的组件。

最简示例：

```
getInitialState: function() {
  var ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
  return {
dataSource: ds.cloneWithRows(['行 1', '行 2']),
  };
},

render: function() {
  return (
    <ListView
dataSource={this.state.dataSource}
      renderRow={(rowData) => <Text>{rowData}</Text>}
    />
  );
},
```

ListView 还支持更高级的功能，包括带有粘性分区标题的分区、头部和尾部支持、达到可用数据末尾时的回调（onEnd Reached）以及设备视口中可见行集合变化时的回调（onChangeVisibleRows），还有若干性能优化。

有一些性能操作旨在使 ListView 在动态加载可能非常大（或概念上无限）的数据集时滚动流畅：

- 仅重新渲染已更改的行——提供给数据源的 rowHasChanged 函数告诉 ListView 是否需要重新渲染某行，因为源数据已更改——详情请参见 ListViewDataSource。
- 限速行渲染——默认情况下，每个事件循环只渲染一行（可通过 pageSize 属性自定义）。这将工作分解为更小的块，以减少渲染行时丢帧的可能性。

Chapter 11: ListView

Section 11.1: Simple Example

ListView - A core component designed for efficient display of vertically scrolling lists of changing data. The minimal API is to create a ListView.DataSource, populate it with a simple array of data blobs, and instantiate a ListView component with that data source and a renderRow callback which takes a blob from the data array and returns a renderable component.

Minimal example:

```
getInitialState: function() {
  var ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
  return {
    dataSource: ds.cloneWithRows(['row 1', 'row 2']),
  };
},

render: function() {
  return (
    <ListView
      dataSource={this.state.dataSource}
      renderRow={(rowData) => <Text>{rowData}</Text>}
    />
  );
},
```

ListView also supports more advanced features, including sections with sticky section headers, header and footer support, callbacks on reaching the end of the available data (onEndReached) and on the set of rows that are visible in the device viewport change (onChangeVisibleRows), and several performance optimizations.

There are a few performance operations designed to make ListView scroll smoothly while dynamically loading potentially very large (or conceptually infinite) data sets:

- Only re-render changed rows - the rowHasChanged function provided to the data source tells the ListView if it needs to re-render a row because the source data has changed - see ListViewDataSource for more details.
- Rate-limited row rendering - By default, only one row is rendered per event-loop (customizable with the pageSize prop). This breaks up the work into smaller chunks to reduce the chance of dropping frames while rendering rows.

第12章：带有ListView的刷新控件

第12.1节：带有ListView的刷新控件完整示例

RefreshControl用于ScrollView或ListView中以添加下拉刷新功能。在本示例中，我们将与ListView一起使用它

```
'use strict'
import React, { Component } from 'react';
import { StyleSheet, View, ListView, RefreshControl, Text } from 'react-native'


class RefreshControlExample extends Component {
  constructor () {
    super()
    this.state = {
refreshing: false,
dataSource: new ListView.DataSource({
        rowHasChanged: (row1, row2) => row1 !== row2 }),
      cars : [
        {name:'达特桑',color:'白色'},
        {name:'凯美瑞',color:'绿色'}
      ]
    }
  }

componentWillMount(){
     this.setState({ dataSource:
       this.state.dataSource.cloneWithRows(this.state.cars) })
   }

render() {
    return (
      <View style={{flex:1}}>
        <ListView
refreshControl={this._refreshControl()}
          dataSource={this.state.dataSource}
renderRow={(car) => this._renderListView(car)}>
        </ListView>
      </View>
    )
  }

_renderListView(car){
    return(
      <View style={styles.listView}>
        <Text>{car.name}</Text>
        <Text>{car.color}</Text>
      </View>
    )
  }

_refreshControl(){
    return (
      <RefreshControl
refreshing={this.state.refreshing}
        onRefresh={()=>this._refreshListView()} />
    )
  }
```

Chapter 12: RefreshControl with ListView

Section 12.1: Refresh Control with ListView Full Example

RefreshControl is used inside a ScrollView or ListView to add pull to refresh functionality. at this example we will use it with ListView

```
'use strict'
import React, { Component } from 'react';
import { StyleSheet, View, ListView, RefreshControl, Text } from 'react-native'


class RefreshControlExample extends Component {
  constructor () {
    super()
    this.state = {
      refreshing: false,
      dataSource: new ListView.DataSource({
        rowHasChanged: (row1, row2) => row1 !== row2 }),
      cars : [
        {name:'Datsun',color:'White'},
        {name:'Camry',color:'Green'}
      ]
    }
  }

   componentWillMount(){
     this.setState({ dataSource:
       this.state.dataSource.cloneWithRows(this.state.cars) })
   }

  render() {
    return (
      <View style={{flex:1}}>
        <ListView
          refreshControl={this._refreshControl()}
          dataSource={this.state.dataSource}
          renderRow={(car) => this._renderListView(car)}>
        </ListView>
      </View>
    )
  }

  _renderListView(car){
    return(
      <View style={styles.listView}>
        <Text>{car.name}</Text>
        <Text>{car.color}</Text>
      </View>
    )
  }

  _refreshControl(){
    return (
      <RefreshControl
        refreshing={this.state.refreshing}
        onRefresh={()=>this._refreshListView()} />
    )
  }
_refreshListView(){
    //开始渲染加载动画
    this.setState({refreshing:true})
    this.state.cars.push(
      {name:'Fusion',color:'Black'},
      {name:'Yaris',color:'Blue'}
    )
    //使用新数据更新数据源
    this.setState({ dataSource:
        this.state.dataSource.cloneWithRows(this.state.cars) })
    this.setState({refreshing:false}) //停止渲染加载动画
  }

}

const styles = StyleSheet.create({

  listView: {
flex: 1,
backgroundColor:'#fff',
    marginTop:10,
marginRight:10,
   marginLeft:10,
   padding:10,
   borderWidth:0.5,
   borderColor:'#dddddd',
   height:70
  }

})

module.exports = RefreshControlExample
```

第12.2节：刷新控件

```
_refreshControl(){
    return (
      <RefreshControl
refreshing={this.state.refreshing}
        onRefresh={()=>this._refreshListView()} />
    )
  }
```

refreshing: 是加载指示器的状态（true，false）。

onRefresh: 当刷新ListView/ScrollView时调用的函数。

第12.3节：onRefresh函数示例

```
_refreshListView(){
    //开始渲染加载动画
    this.setState({refreshing:true})
    this.state.cars.push(
      {name:'Fusion',color:'Black'},
      {name:'Yaris',color:'Blue'}
    )
    //使用新数据更新数据源
    this.setState({ dataSource:
        this.state.dataSource.cloneWithRows(this.state.cars) })
    this.setState({refreshing:false}) //停止渲染加载指示器
```

```
  _refreshListView(){
    //Start Rendering Spinner
    this.setState({refreshing:true})
    this.state.cars.push(
      {name:'Fusion',color:'Black'},
      {name:'Yaris',color:'Blue'}
    )
    //Updating the dataSource with new data
    this.setState({ dataSource:
        this.state.dataSource.cloneWithRows(this.state.cars) })
    this.setState({refreshing:false}) //Stop Rendering Spinner
  }

}

const styles = StyleSheet.create({

  listView: {
    flex: 1,
    backgroundColor:'#fff',
    marginTop:10,
    marginRight:10,
    marginLeft:10,
    padding:10,
    borderWidth:.5,
    borderColor:'#dddddd',
    height:70
  }

})

module.exports = RefreshControlExample
```

Section 12.2: Refresh Control

```
  _refreshControl(){
    return (
      <RefreshControl
        refreshing={this.state.refreshing}
        onRefresh={()=>this._refreshListView()} />
    )
  }
```

refreshing: is the state of the spinner (true, false).

onRefresh: this function will invoke when refresh the ListView/ScrollView.

Section 12.3: onRefresh function Example

```
  _refreshListView(){
    //Start Rendering Spinner
    this.setState({refreshing:true})
    this.state.cars.push(
      {name:'Fusion',color:'Black'},
      {name:'Yaris',color:'Blue'}
    )
    //Updating the dataSource with new data
    this.setState({ dataSource:
        this.state.dataSource.cloneWithRows(this.state.cars) })
    this.setState({refreshing:false}) //Stop Rendering Spinner
  }
```

这里我们更新数组，之后更新dataSource。我们可以使用fetch从服务器请求数据，并使用async/await。

```
  }
```

here we are updating the array and after that we will update the dataSource. we can use fetch to request something from server and use async/await.

第13章：WebView

WebView可用于加载外部网页或HTML内容。该组件默认存在。

第13.1节：使用webview的简单组件

```
import React, { Component } from 'react';
import { WebView } from 'react-native';

class MyWeb extends Component {
render() {
    return (
      <WebView
source={{uri: 'https://github.com/facebook/react-native'}}
        style={{marginTop: 20}}
       />
    );
  }
}
```

Chapter 13: WebView

Webview can be used to load external webpages or html content. This component is there by default.

Section 13.1: Simple component using webview

```
import React, { Component } from 'react';
import { WebView } from 'react-native';

class MyWeb extends Component {
  render() {
    return (
      <WebView
        source={{uri: 'https://github.com/facebook/react-native'}}
        style={{marginTop: 20}}
      />
    );
  }
}
```

第14章：命令行指令

第14.1节：检查已安装版本

```
$ react-native -v
```

示例输出

```
react-native-cli: 0.2.0
react-native: n/a - 不在 React Native 项目目录内 // 来自不同文件夹的输出
react-native: react-native: 0.30.0 // 来自 React Native 项目目录的输出
```

第14.2节：初始化并开始使用React Native项目

初始化方法

```
react-native init MyAwesomeProject
```

使用指定版本的React Native进行初始化

```
react-native init --version="0.36.0" MyAwesomeProject
```

运行Android

```
cd MyAwesomeProject
react-native run-android
```

运行iOS

```
cd MyAwesomeProject
react-native run-ios
```

第14.3节：将现有项目升级到最新的React Native版本

在 app 文件夹中找到`package.json`并修改以下行以包含最新版本，保存文件并关闭。

```
"react-native": "0.32.0"
```

在终端：

```
$ npm install
```

接着执行

```
$ react-native upgrade
```

第14.4节：为您的应用添加安卓项目

如果您之前生成的应用没有安卓支持，或者您是故意这样做的，您总是可以为您的应用添加安卓项目。

Chapter 14: Command Line Instructions

Section 14.1: Check version installed

```
$ react-native -v
```

Example Output

```
react-native-cli: 0.2.0
react-native: n/a - not inside a React Native project directory //Output from  different folder
react-native: react-native: 0.30.0 // Output from the react native project directory
```

Section 14.2: Initialize and getting started with React Native project

To initialize

```
react-native init MyAwesomeProject
```

To initialize with a specific version of React Native

```
react-native init --version="0.36.0" MyAwesomeProject
```

To Run for Android

```
cd MyAwesomeProject
react-native run-android
```

To Run for iOS

```
cd MyAwesomeProject
react-native run-ios
```

Section 14.3: Upgrade existing project to latest RN version

In the app folder find `package.json` and modify the following line to include the latest version, save the file and close.

```
"react-native": "0.32.0"
```

In terminal:

```
$ npm install
```

Followed by

```
$ react-native upgrade
```

Section 14.4: Add android project for your app

If you either have apps generated with pre-android support or just did that on purpose, you can always add android project to your app.

```
$ react-native android
```

这将会在您的应用中生成android文件夹和index.android.js文件。

第14.5节：日志记录

Android

```
$ react-native log-android
```

iOS

```
$ react-native log-ios
```

第14.6节：启动React Native打包器

```
$ react-native start
```

在最新版本的React Native中，无需手动运行打包器。它会自动运行。

默认情况下，这会在端口8081启动服务器。若要指定服务器端口

```
$ react-native start --port PORTNUMBER
```

```
$ react-native android
```

This will generate `android` folder and `index.android.js` inside your app.

Section 14.5: Logging

Android

```
$ react-native log-android
```

iOS

```
$ react-native log-ios
```

Section 14.6: Start React Native Packager

```
$ react-native start
```

On latest version of React Native, no need to run the packager. It will run automatically.

By default this starts the server at port 8081. To specify which port the server is on

```
$ react-native start --port PORTNUMBER
```

第15章：HTTP请求

第15.1节：在fetch API和Redux中使用Promise

Redux是React-Native中最常用的状态管理库。以下示例演示了如何使用fetch API并通过redux-thunk分发更改到应用程序的状态reducer。

```
export const fetchRecipes = (action) => {
  return (dispatch, getState) => {
fetch('/recipes', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
body: JSON.stringify({
          recipeName,
instructions,
        ingredients
        })
    })
.then((res) => {
      // 如果响应成功，则解析json并分发更新
      if (res.ok) {
res.json().then((recipe) => {
          dispatch({
类型: 'UPDATE_RECIPE',
          配方
          });
        });
      } 否则 {
        // 响应不成功，因此派发错误
       res.json().then((err) => {
        dispatch({
类型: 'ERROR_RECIPE',
          信息: err.原因,
          状态: err.状态
          });
        });
      }
    })
.捕获((err) => {
      // 如果发生一般的JavaScript错误，则执行此处。
dispatch(错误('请求出现问题。'));
    });
  };
};
```

第15.2节：使用fetch API的HTTP

需要注意的是，Fetch 不支持进度回调。详见：https://github.com/github/fetch/issues/89。

另一种方法是使用 XMLHttpRequest https://developer.mozilla.org/en-US/docs/Web/Events/progress。

```
fetch('https://mywebsite.com/mydata.json').then(json => console.log(json));

fetch('/login', {
method: 'POST',
```

Chapter 15: HTTP Requests

Section 15.1: Using Promises with the fetch API and Redux

Redux is the most common state management library used with React-Native. The following example demonstrates how to use the fetch API and dispatch changes to your applications state reducer using redux-thunk.

```
export const fetchRecipes = (action) => {
  return (dispatch, getState) => {
    fetch('/recipes', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          recipeName,
          instructions,
          ingredients
        })
    })
    .then((res) => {
      // If response was successful parse the json and dispatch an update
      if (res.ok) {
        res.json().then((recipe) => {
          dispatch({
            type: 'UPDATE_RECIPE',
            recipe
          });
        });
      } else {
        // response wasn't successful so dispatch an error
        res.json().then((err) => {
          dispatch({
            type: 'ERROR_RECIPE',
            message: err.reason,
            status: err.status
          });
        });
      }
    })
    .catch((err) => {
      // Runs if there is a general JavaScript error.
      dispatch(error('There was a problem with the request.'));
    });
  };
};
```

Section 15.2: HTTP with the fetch API

It should be noted that Fetch *does not support progress callbacks*. See: https://github.com/github/fetch/issues/89.

The alternative is to use XMLHttpRequest https://developer.mozilla.org/en-US/docs/Web/Events/progress.

```
fetch('https://mywebsite.com/mydata.json').then(json => console.log(json));

fetch('/login', {
  method: 'POST',
body: form,
mode: 'cors',
cache: 'default',
}).then(session => onLogin(session), failure => console.error(failure));
```

关于 fetch 的更多详情可以在 MDN找到

第 15.3 节：使用 XMLHttpRequest 进行网络通信

```
var request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
   if (request.readyState !== 4) {
      return;
   }

   if (request.status === 200) {
     console.log('成功', request.responseText);
   } else {
console.warn('错误');
   }
};

request.open('GET', 'https://mywebsite.com/endpoint/');
request.send();
```

第15.4节：WebSockets

```
var ws = new WebSocket('ws://host.com/path');

ws.onopen = () => {
   // 连接已打开

ws.send('某些内容'); // 发送消息
};

ws.onmessage = (e) => {
   // 收到一条消息
console.log(e.data);
};

ws.onerror = (e) => {
   // 发生错误
console.log(e.message);
};

ws.onclose = (e) => {
   // 连接关闭
console.log(e.code, e.reason);
};
```

第15.5节：使用axios的Http

配置

对于网络请求，你也可以使用库axios。

配置很简单。为此你可以创建一个名为 axios.js 的文件，例如：

```
  body: form,
  mode: 'cors',
  cache: 'default',
}).then(session => onLogin(session), failure => console.error(failure));
```

More details about fetch can be found at MDN

Section 15.3: Networking with XMLHttpRequest

```
var request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
  if (request.readyState !== 4) {
    return;
  }

  if (request.status === 200) {
    console.log('success', request.responseText);
  } else {
    console.warn('error');
  }
};

request.open('GET', 'https://mywebsite.com/endpoint/');
request.send();
```

Section 15.4: WebSockets

```
var ws = new WebSocket('ws://host.com/path');

ws.onopen = () => {
  // connection opened

  ws.send('something'); // send a message
};

ws.onmessage = (e) => {
  // a message was received
  console.log(e.data);
};

ws.onerror = (e) => {
  // an error occurred
  console.log(e.message);
};

ws.onclose = (e) => {
  // connection closed
  console.log(e.code, e.reason);
};
```

Section 15.5: Http with axios

Configure

For web request you can also use library axios.

It's easy to configure. For this purpose you can create file axios.js for example:

```
import * as axios from 'axios';

var instance = axios.create();
instance.defaults.baseURL = serverURL;
instance.defaults.timeout = 20000;]
//...
//以及其他选项

export { instance as default };
```

然后你可以在任何你想要的文件中使用它。

请求

为了避免在后端的每个服务中都使用“瑞士军刀”模式，你可以在集成功能的文件夹中创建一个包含这些方法的单独文件：

```
import axios from '../axios';
import {
errorHandling
} from '../common';

const UserService = {
getCallToAction() {
        return axios.get('api/user/dosomething').then(response => response.data)
            .catch(errorHandling);
    },
}
export default UserService;
```

Testing

有一个专门用于测试 axios 的库：axios-mock-adapter。

使用这个库，你可以为 axios 设置任何你想要的响应以进行测试。你还可以为你的 axios 方法配置一些特殊的错误。你可以将它添加到之前步骤中创建的 axios.js 文件中：

```
import MockAdapter from 'axios-mock-adapter';

var mock = new MockAdapter(instance);
mock.onAny().reply(500);
```

例如。

Redux 存储

有时你需要在请求头中添加授权令牌，这个令牌可能存储在你的 Redux 存储中。

在这种情况下，你需要另一个文件 interceptors.js，里面包含这个函数：

```
export function getAuthToken(storeContainer) {
    return config => {
        let store = storeContainer.getState();
        config.headers['Authorization'] = store.user.accessToken;
        return config;
    };
}
```

```
import * as axios from 'axios';

var instance = axios.create();
instance.defaults.baseURL = serverURL;
instance.defaults.timeout = 20000;]
//...
//and other options

export { instance as default };
```

and then use it in any file you want.

Requests

To avoid using pattern 'Swiss knife' for every service on your backend you can create separate file with methods for this within folder for integration functionality:

```
import axios from '../axios';
import {
    errorHandling
} from '../common';

const UserService = {
        getCallToAction() {
        return axios.get('api/user/dosomething').then(response => response.data)
            .catch(errorHandling);
    },
}
export default UserService;
```

Testing

There is a special lib for testing axios: axios-mock-adapter.

With this lib you can set to axios any responce you want for testing it. Also you can configure some special errors for your axois'es methods. You can add it to your axios.js file created in prevous step:

```
import MockAdapter from 'axios-mock-adapter';

var mock = new MockAdapter(instance);
mock.onAny().reply(500);
```

for example.

Redux Store

Sometimes you need to add to headers authorize token, that you probably store in your redux store.

In this case you'll need another file, interceptors.js with this function:

```
export function getAuthToken(storeContainer) {
    return config => {
        let store = storeContainer.getState();
        config.headers['Authorization'] = store.user.accessToken;
        return config;
    };
}
```

接下来，在你的根组件的构造函数中可以添加以下代码：

```
axios.interceptors.request.use(getAuthToken(this.state.store));
```

这样你的所有请求都会携带你的授权令牌。

正如你所见，axios 是一个非常简单、可配置且对基于 react-native 的应用程序非常有用的库。

第15.6节：使用 Socket.io 的 Web Socket

安装 *socket.io-client*

```
npm i socket.io-client --save
```

导入模块

```
import SocketIOClient from 'socket.io-client/dist/socket.io.js'
```

在构造函数中初始化

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');
  }
```

现在，为了正确使用你的 socket 连接，你也应该在构造函数中绑定你的函数。假设我们需要构建一个简单的应用程序，该应用程序将每隔5秒通过 socket 向服务器发送一次 ping（将其视为 ping），然后应用程序将从服务器接收回复。为此，先创建这两个函数：

```
_sendPing(){
    //向socket服务器发送ding消息
socket.emit('ding');
}

_getReply(data){
    //从socket服务器获取回复，并打印到控制台
console.log('来自服务器的回复:' + data);
}
```

现在，我们需要在构造函数中绑定这两个函数：

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');

    //绑定函数
    this._sendPing = this._sendPing.bind(this);
    this._getReply = this._getReply.bind(this);
}
```

之后，我们还需要将_getReply函数与socket关联，以便接收来自socket服务器的消息。为此，我们需要将_getReply函数附加到socket对象。请在构造函数中添加以下代码：

```
this.socket.on('dong', this._getReply);
```

Next in constructor of your root component you can add this:

```
axios.interceptors.request.use(getAuthToken(this.state.store));
```

and then all your requests will be followed with your authorization token.

As you can see axios is very simple, configurable and useful library for applications based on react-native.

Section 15.6: Web Socket with Socket.io

Install *socket.io-client*

```
npm i socket.io-client --save
```

Import module

```
import SocketIOClient from 'socket.io-client/dist/socket.io.js'
```

Initialize in your constructor

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');
  }
```

Now in order to use your socket connection properly, you should bind your functions in constructor too. Let's assume that we have to build a simple application, which will send a ping to a server via socket after every 5 seconds (consider this as ping), and then the application will get a reply from the server. To do so, let's first create these two functions:

```
_sendPing(){
    //emit a dong message to socket server
    socket.emit('ding');
}

_getReply(data){
    //get reply from socket server, log it to console
    console.log('Reply from server:' + data);
}
```

Now, we need to bind these two functions in our constructor:

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');

    //bind the functions
    this._sendPing = this._sendPing.bind(this);
    this._getReply = this._getReply.bind(this);
}
```

After that, we also need to link _getReply function with the socket in order to receive the message from the socket server. To do this we need to attach our _getReply function with socket object. Add the following line to our constructor:

```
this.socket.on('dong', this._getReply);
```

现在，每当 socket 服务器发出 'dong' 事件时，你的应用程序都能接收到它。

Now, whenever socket server emits with the 'dong' your application will able to receive it.

第16章：平台模块

第16.1节：查找操作系统类型/版本

第一步是从 'react-native' 包中导入 Platform，如下所示：

```
import { Platform } from 'react-native'
```

完成后，你就可以通过 Platform.OS 访问操作系统类型，从而在条件语句中使用它，例如

```
const styles = StyleSheet.create({
  height: (Platform.OS === 'ios') ? 200 : 100,
})
```

如果你想检测 Android 版本，可以使用 Platform.Version，方法如下：

```
if (Platform.Version === 21) {
  console.log('正在运行 Lollipop！');
}
```

对于 iOS，Platform.Version 返回的是字符串，对于复杂条件不要忘记解析它。

```
if (parseInt(Platform.Version, 10) >= 9) {
    console.log('运行版本高于 8');
}
```

如果平台特定的逻辑较复杂，可以根据平台渲染两个不同的文件。例如：

- `MyTask.android.js`
- `MyTask.ios.js`

并使用以下方式引入

```
const MyTask = require('./MyTask')
```

Chapter 16: Platform Module

Section 16.1: Find the OS Type/Version

The first step is to import Platform from the 'react-native' package like so:

```
import { Platform } from 'react-native'
```

After you've done that, you can go ahead and access the OS type through `Platform.OS` allowing you to use it in conditional statements like

```
const styles = StyleSheet.create({
  height: (Platform.OS === 'ios') ? 200 : 100,
})
```

If you want to detect the Android version, you can use `Platform.Version` like so:

```
if (Platform.Version === 21) {
  console.log('Running on Lollipop!');
}
```

For iOS, Platform.Version is returning a String, for complex condition don't forget to parse it.

```
if (parseInt(Platform.Version, 10) >= 9) {
    console.log('Running version higher than 8');
}
```

If the platform specific logic is complex, one can render two different files based on platform. Ex:

- `MyTask.android.js`
- `MyTask.ios.js`

and require it using

```
const MyTask = require('./MyTask')
```

第17章：图像

第17.1节：图像模块

你需要从 react-native 包中导入 Image，然后使用它：

```
import { Image } from 'react';

<Image source={{uri: 'https://image-souce.com/awesomeImage'}} />
```

你也可以使用本地图片，语法稍有不同但逻辑相同，如下所示：

```
import { Image } from 'react';

<Image source={require('./img/myCoolImage.png')} />
```

注意：你应该给图片设置高度和宽度，否则图片不会显示。

第17.2节：图片示例

```
class ImageExample extends Component {
  render() {
    return (
      <View>
        <Image style={{width: 30, height: 30}}
          source={{uri: 'http://facebook.github.io/react/img/logo_og.png'}}
        />
      </View>
    );
  }
}
```

第17.3节：条件图片来源

```
<Image style={[this.props.imageStyle]}
       source={this.props.imagePath
      ? this.props.imagePath
       : require('../theme/images/resource.png')}
   />
```

如果路径存在于imagePath中，则将其分配给source，否则将分配默认的图片路径。

第17.4节：使用变量存储图片路径

```
let imagePath = require("../../assets/list.png");

<Image style={{height: 50, width: 50}} source={imagePath} />
```

来自外部资源：

```
<Image style={{height: 50, width: 50}} source={{uri: userData.image}} />
```

Chapter 17: Images

Section 17.1: Image Module

You're going to have to import `Image` from the `react-native` package like so then use it:

```
import { Image } from 'react';

<Image source={{uri: 'https://image-souce.com/awesomeImage'}} />
```

You can also use a local image with a slightly different syntax but same logic like so:

```
import { Image } from 'react';

<Image source={require('./img/myCoolImage.png')} />
```

Note: You should give height, width to the image otherwise it won't show.

Section 17.2: Image Example

```
class ImageExample extends Component {
  render() {
    return (
      <View>
        <Image style={{width: 30, height: 30}}
          source={{uri: 'http://facebook.github.io/react/img/logo_og.png'}}
        />
      </View>
    );
  }
}
```

Section 17.3: Conditional Image Source

```
<Image style={[this.props.imageStyle]}
       source={this.props.imagePath
       ? this.props.imagePath
       : require('../theme/images/resource.png')}
   />
```

If the path is available in `imagePath` then it will be assigned to source else the default image path will be assigned.

Section 17.4: Using variable for image path

```
let imagePath = require("../../assets/list.png");

<Image style={{height: 50, width: 50}} source={imagePath} />
```

From external resource:

```
<Image style={{height: 50, width: 50}} source={{uri: userData.image}} />
```

第17.5节：调整图片大小以适应

```
<Image
    resizeMode="contain"
    style={{height: 100, width: 100}}
    source={require('../assets/image.png')} />
```

也可以尝试cover、stretch、repeat和center参数。

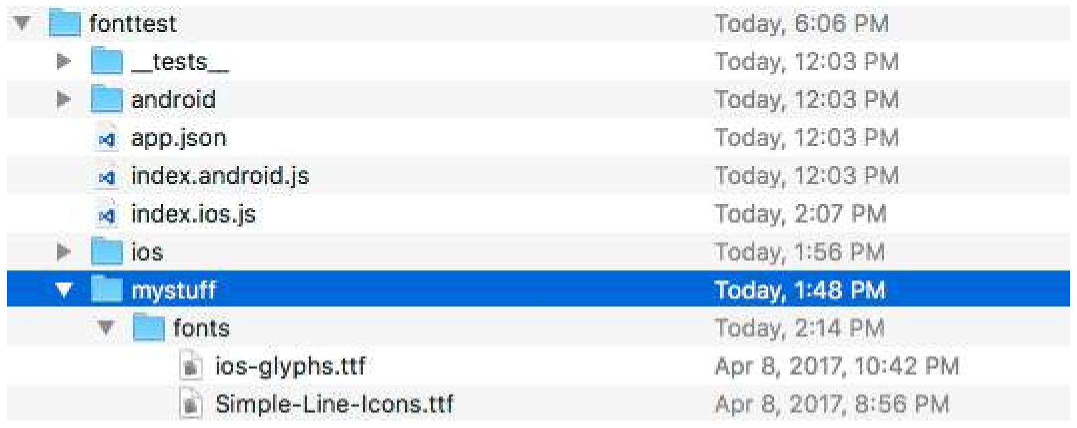
Section 17.5: To fit an Image

```
<Image
    resizeMode="contain"
    style={{height: 100, width: 100}}
    source={require('../assets/image.png')} />
```

Try also **cover**, **stretch**, **repeat** and **center** parameters.

第18章：自定义字体

第18.1节：适用于Android和iOS的自定义字体

- 在你的项目文件夹中创建一个文件夹，并将字体添加到其中。例如：
  - 示例：这里我们在根目录下添加了一个名为“mystuff”的文件夹，然后是“fonts”，并在其中放置了我们的字体：
    - 

```
▼ fonttest                      Today, 6:06 PM
  ▶ __tests__                   Today, 12:03 PM
  ▶ android                     Today, 12:03 PM
    app.json                    Today, 12:03 PM
    index.android.js            Today, 12:03 PM
    index.ios.js                Today, 2:07 PM
  ▶ ios                         Today, 1:56 PM
  ▼ mystuff                     Today, 1:48 PM
    ▼ fonts                     Today, 2:14 PM
        ios-glyphs.ttf          Apr 8, 2017, 10:42 PM
        Simple-Line-Icons.ttf   Apr 8, 2017, 8:56 PM
```

- 在package.json中添加以下代码。

{ ... "rnpm": { "assets": ["path/to/fontfolder"] }, ... }
  - 以上述示例为例，我们的package.json现在将包含路径“mystuff/fonts”：

```
"rnpm": {
  "assets": [
    "mystuff/fonts"
  ]
}
```

- 运行react-native link命令。
- 在项目中使用自定义字体的以下代码

<Text style={{ fontFamily: 'FONT-NAME' }}> 我的文本 </Text>

其中`FONT-NAME`是特定平台的前缀。

Android

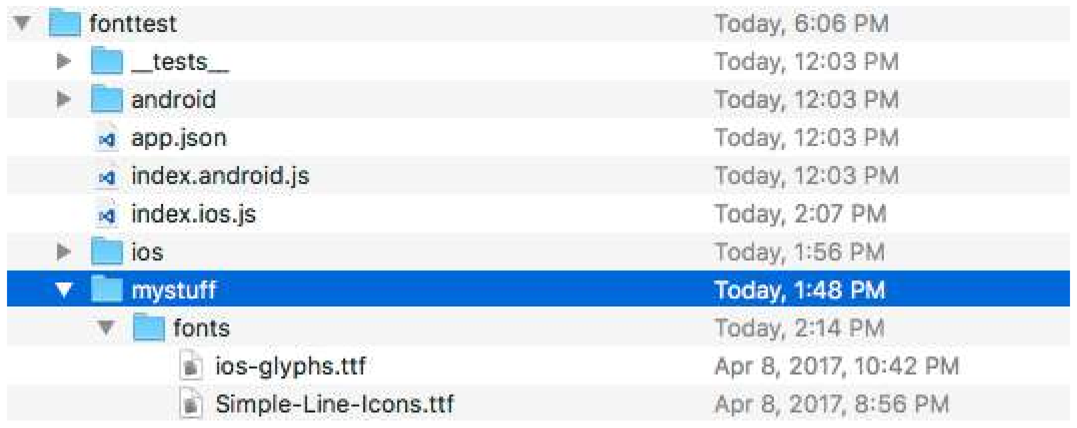
FONT-NAME 是文件扩展名前的文字。例如：您的字体文件名是 `Roboto-Regular.tf`，因此您应设置 `fontFamily: Roboto-Regular`。

iOS

FONT-NAME 是在字体文件上右键点击后，选择“获取信息”中找到的“全名”。（来源：https://stackoverflow.com/a/16788493/2529614），在下面的截图中，文件名是`MM Proxima Nova Ultra bold.otf`，然而“全名”是“Proxima Nova Semibold”，因此你应设置`fontFamily: Proxima Nova Semibold`。截图 -

Chapter 18: Custom Fonts

Section 18.1: Custom fonts for both Android and IOS

- Create a folder in your project folder, and add your fonts to it. Example:
  - Example: Here we added a folder in root called "mystuff", then "fonts", and inside it we placed our fonts:
    - 

```
▼ fonttest                      Today, 6:06 PM
  ▶ __tests__                   Today, 12:03 PM
  ▶ android                     Today, 12:03 PM
    app.json                    Today, 12:03 PM
    index.android.js            Today, 12:03 PM
    index.ios.js                Today, 2:07 PM
  ▶ ios                         Today, 1:56 PM
  ▼ mystuff                     Today, 1:48 PM
    ▼ fonts                     Today, 2:14 PM
        ios-glyphs.ttf          Apr 8, 2017, 10:42 PM
        Simple-Line-Icons.ttf   Apr 8, 2017, 8:56 PM
```

- Add the below code in `package.json`.

{ ... "rnpm": { "assets": ["path/to/fontfolder"] }, ... }
  - For the example above, our package.json would now have a path of "mystuff/fonts":

```
"rnpm": {
  "assets": [
    "mystuff/fonts"
  ]
}
```

- Run `react-native link` command.
- Using custom fonts on project below code

<Text style={{ fontFamily: 'FONT-NAME' }}> My Text </Text>

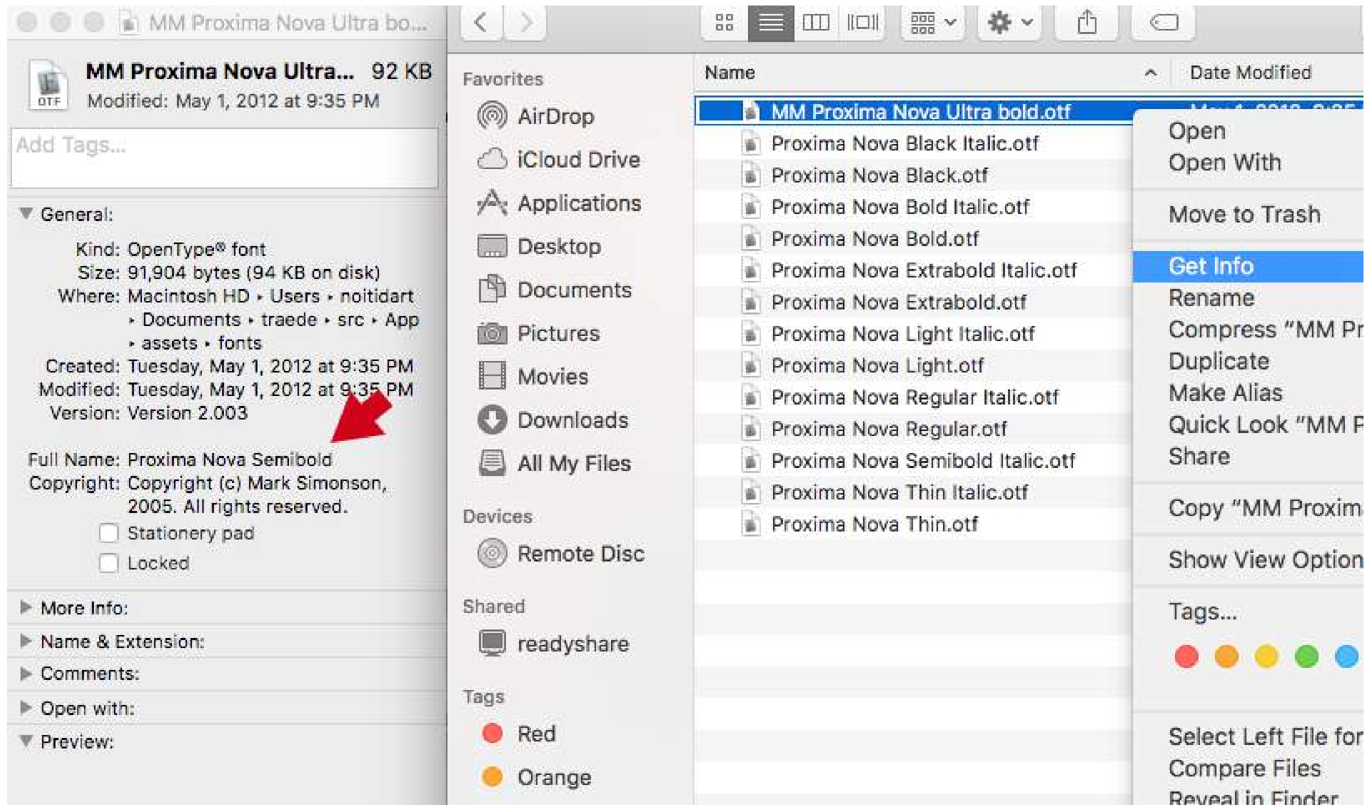
Where `FONT-NAME` is the prefix platform specific.

Android

FONT-NAME is the words before the extension in file. Example: Your font's file name is `Roboto-Regular.ttf`, so you would set `fontFamily: Roboto-Regular`.

iOS

FONT-NAME is "Full Name" found after right clicking, on the font file, then clicking on "Get Info". (Source: https://stackoverflow.com/a/16788493/2529614), in the screenshot below, the file name is `MM Proxima Nova Ultra bold.otf`, however "Full Name" is "Proxima Nova Semibold", thus you would set `fontFamily: Proxima Nova Semibold`. Screenshot -



MM Proxima Nova Ultra bold.otf 92 KB
Modified: May 1, 2012 at 9:35 PM

General:
Kind: OpenType® font
Size: 91,904 bytes (94 KB on disk)
Where: Macintosh HD ▸ Users ▸ noitidart ▸ Documents ▸ traede ▸ src ▸ App ▸ assets ▸ fonts
Created: Tuesday, May 1, 2012 at 9:35 PM
Modified: Tuesday, May 1, 2012 at 9:35 PM
Version: Version 2.003

Full Name: Proxima Nova Semibold
Copyright: Copyright (c) Mark Simonson, 2005. All rights reserved.

MM Proxima Nova Ultra bold.otf
Proxima Nova Black Italic.otf
Proxima Nova Black.otf
Proxima Nova Bold Italic.otf
Proxima Nova Bold.otf
Proxima Nova Extrabold Italic.otf
Proxima Nova Extrabold.otf
Proxima Nova Light Italic.otf
Proxima Nova Light.otf
Proxima Nova Regular Italic.otf
Proxima Nova Regular.otf
Proxima Nova Semibold Italic.otf
Proxima Nova Thin Italic.otf
Proxima Nova Thin.otf

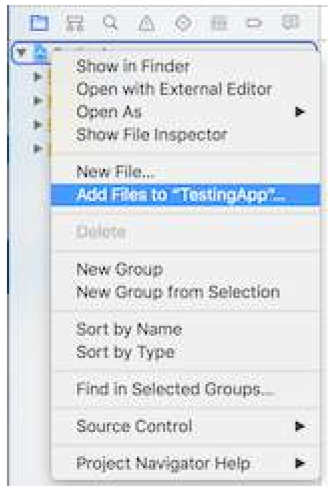
- 再次运行react-native run-ios或react-native run-android（这将重新编译资源）

第18.2节：在React Native中使用自定义字体的步骤（Android）

1. 将你的字体文件粘贴到android/app/src/main/assets/fonts/font_name.ttf
2. 通过运行react-native run-android重新编译Android应用
3. 现在，你可以在React Native样式中使用fontFamily: 'font_name'

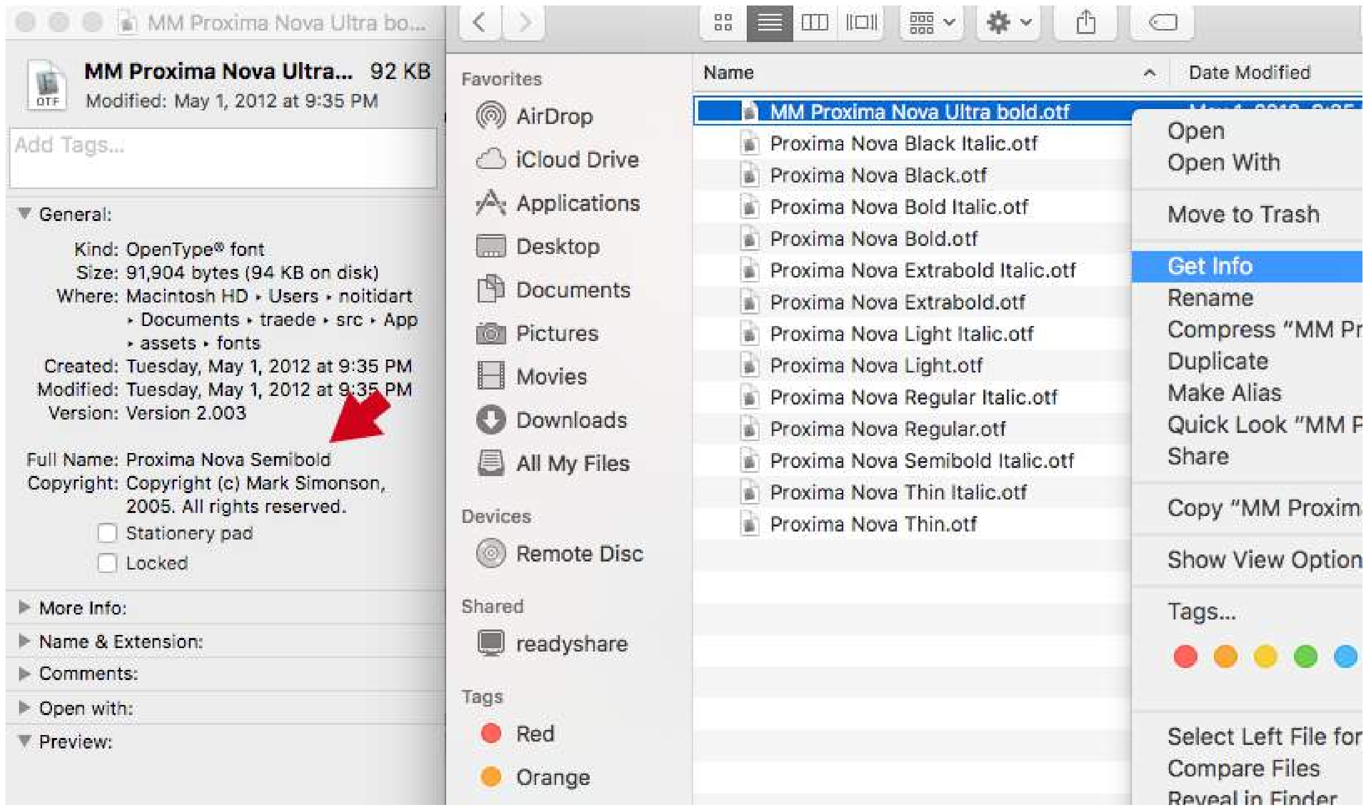
第18.3节：在React Native（iOS）中使用自定义字体的步骤

1. 将字体包含到你的Xcode项目中。



2. 确保它们被包含在目标成员（Target Membership）列中

点击导航器中的字体，检查字体是否被包含。


MM Proxima Nova Ultra bold.otf 92 KB
Modified: May 1, 2012 at 9:35 PM

General:
Kind: OpenType® font
Size: 91,904 bytes (94 KB on disk)
Where: Macintosh HD ▸ Users ▸ noitidart ▸ Documents ▸ traede ▸ src ▸ App ▸ assets ▸ fonts
Created: Tuesday, May 1, 2012 at 9:35 PM
Modified: Tuesday, May 1, 2012 at 9:35 PM
Version: Version 2.003

Full Name: Proxima Nova Semibold
Copyright: Copyright (c) Mark Simonson, 2005. All rights reserved.

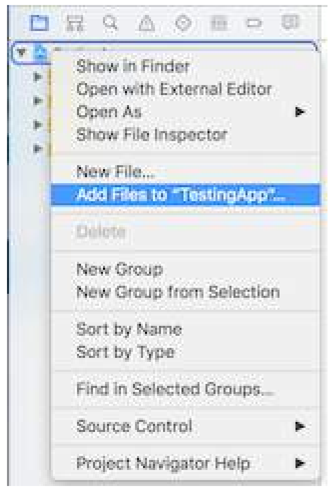
- Run `react-native run-ios` or `react-native run-android` again (this will recompile with the resources)

Section 18.2: Steps to use custom fonts in React Native (Android)

1. Paste your fonts file inside `android/app/src/main/assets/fonts/font_name.ttf`
2. Recompile the Android app by running `react-native run-android`
3. Now, You can use `fontFamily: 'font_name'` in your React Native Styles

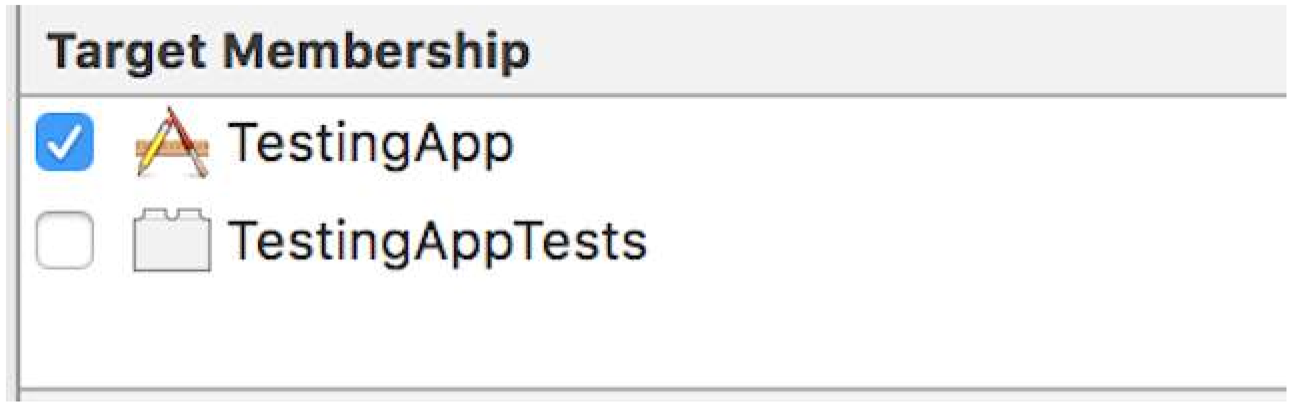
Section 18.3: Steps to use custom fonts in React Native (iOS)

1. Include the font in your Xcode project.



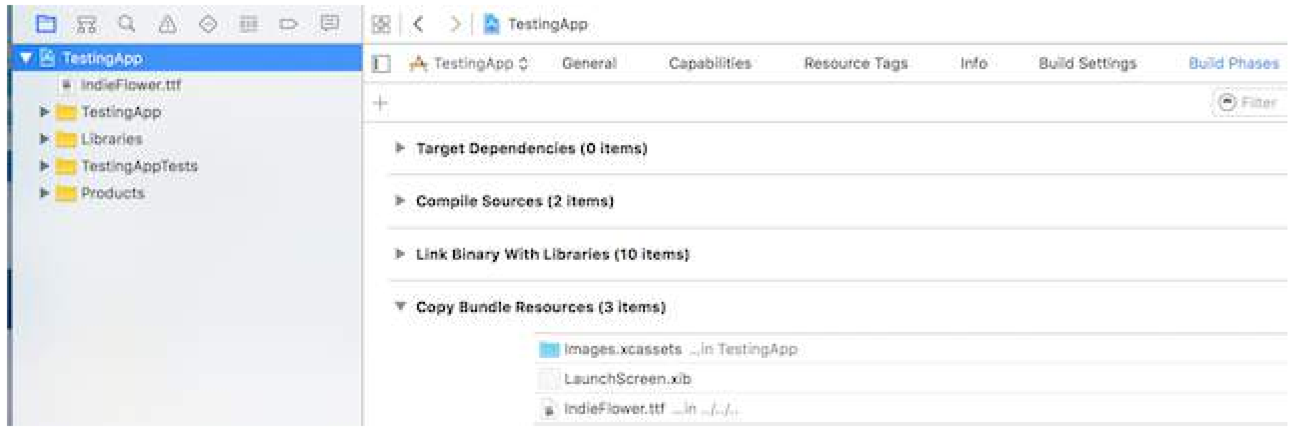
2. Make sure that they are included in the Target Membership column

Click on the font from the navigator, and check if the font included.



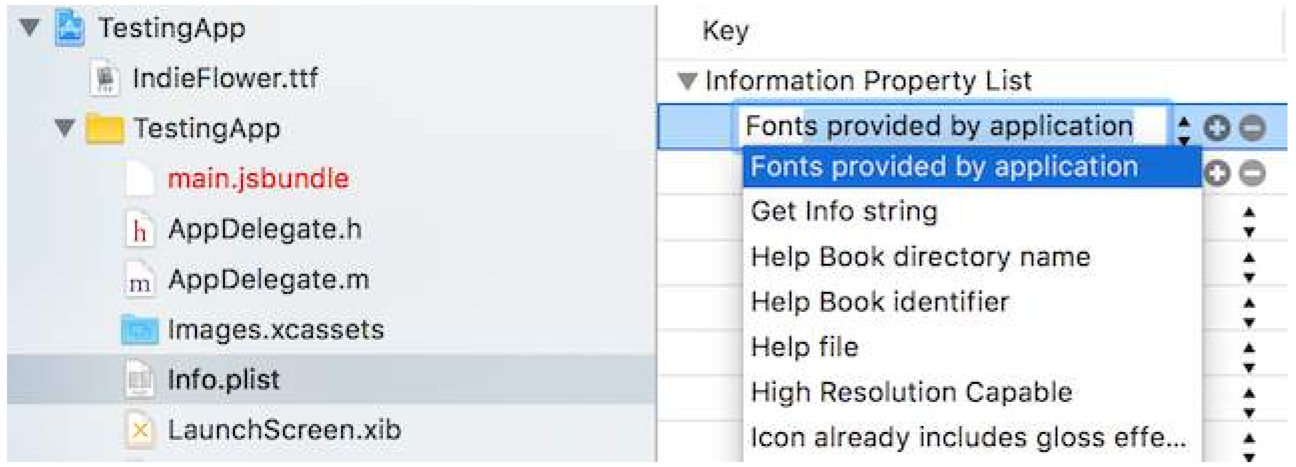
3. 检查字体是否作为资源包含在你的包中

点击你的Xcode项目文件，选择“Build Phases”，再选择“Copy Bundle Resources”。检查你的字体是否已添加。


TestingApp
IndieFlower.ttf
TestingApp
Libraries
TestingAppTests
Products
TestingApp
General
Capabilities
Resource Tags
Info
Build Settings
Build Phases
Filter
Target Dependencies (0 items)
Compile Sources (2 items)
Link Binary With Libraries (10 items)
Copy Bundle Resources (3 items)
Images.xcassets ...in TestingApp
LaunchScreen.xib
IndieFlower.ttf ...in ../../..

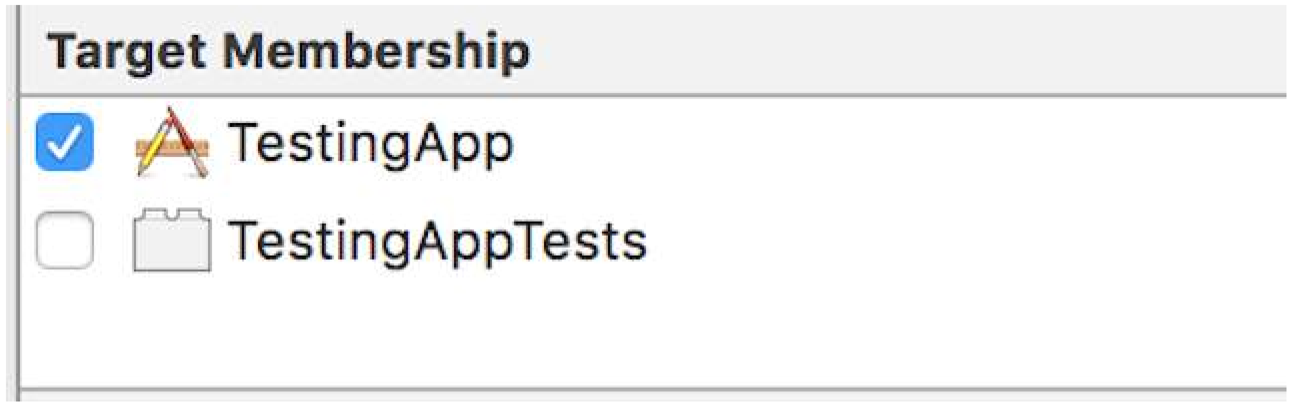
4. 在应用程序的Plist文件（Info.plist）中包含字体

在应用程序主文件夹中打开Info.plist，点击“Information Property List”，然后点击加号（+）。从下拉列表中选择“Fonts provided by application”。



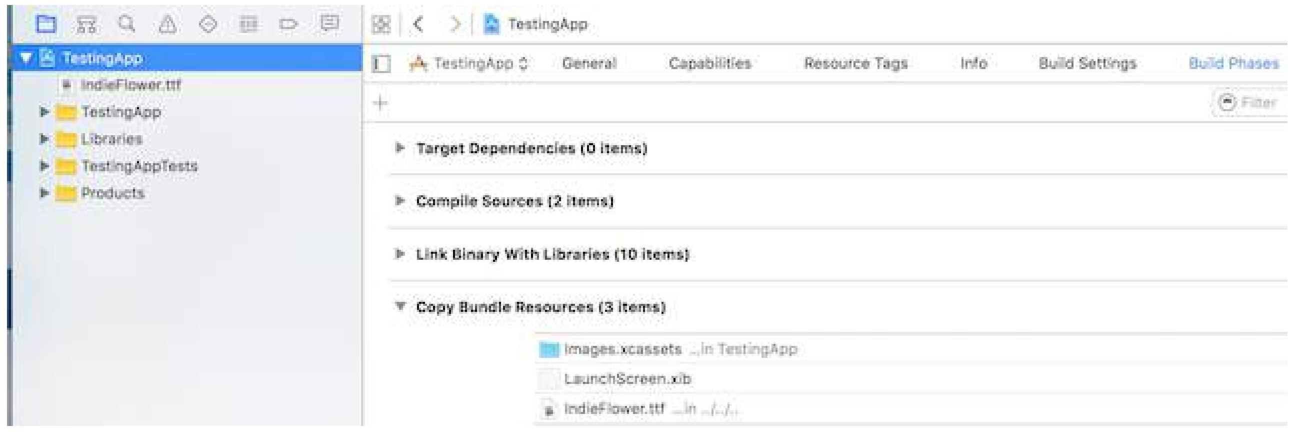
5. 在应用程序提供的字体中添加字体名称

展开应用程序提供的字体，并将字体名称准确添加到值列



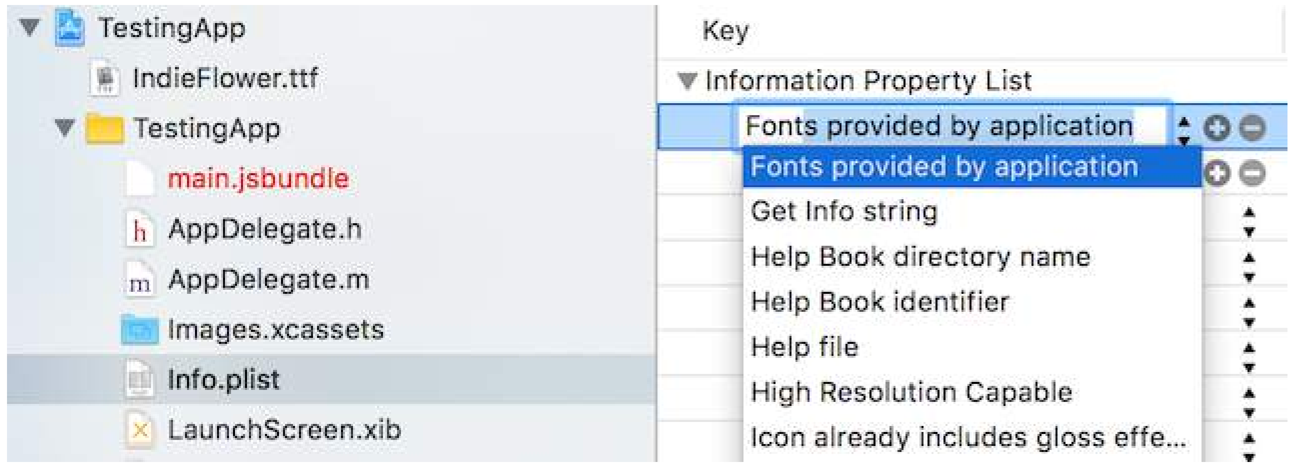
3. Check if the font included as Resource in your bundle

click on your Xcode project file, select "Build Phases, select "Copy Bundle Resources". Check if your font is added.



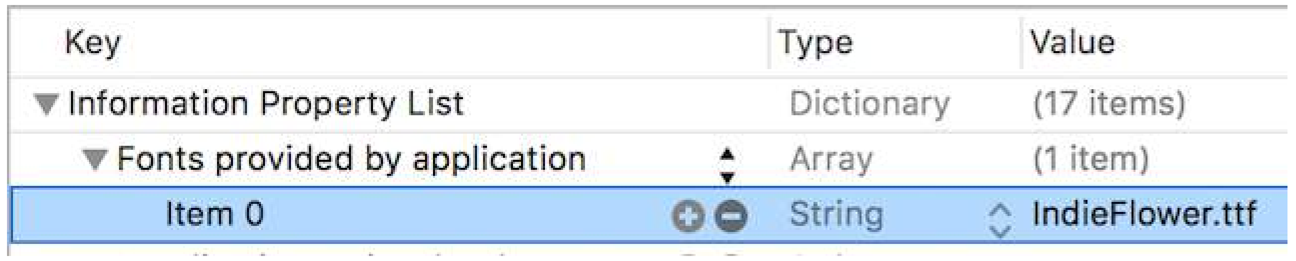
4. Include the font in Application Plist (Info.plist)

from the application main folder open Info.plist, click on "Information Property List", and then click the plus sign (+). from drop down list choose "Fonts provided by application".



5. Add Font name in Fonts provided by application

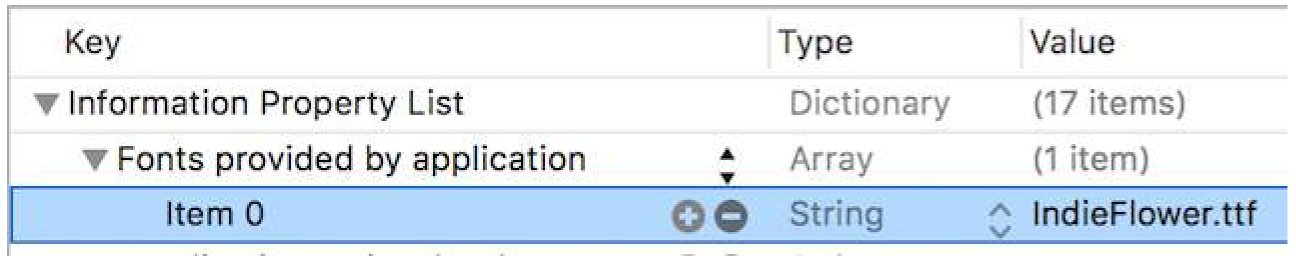
expand Fonts Provided by Application and add the Font Name exactly to value column



Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
▼ Fonts provided by application	Array	(1 item)
Item 0	String	IndieFlower.ttf

6. 在应用程序中使用它

```
<Text style={{fontFamily:'IndieFlower'}}>
  欢迎使用 React Native！
</Text>
```



Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
▼ Fonts provided by application	Array	(1 item)
Item 0	String	IndieFlower.ttf

6. Use it in the Application

```
<Text style={{fontFamily:'IndieFlower'}}>
  Welcome to React Native!
</Text>
```

第19章：动画API

第19.1节：动画图片

```
class AnimatedImage extends Component {
    constructor(props){
        super(props)
        this.state = {
logoMarginTop: new Animated.Value(200)
        }
    }
componentDidMount(){
        Animated.timing(
            this.state.logoMarginTop,
            { toValue: 100 }
        ).start()
    }
render () {
      return (
        <View>
           <Animated.Image source={require('../images/Logo.png')} style={[baseStyles.logo, {
             marginTop: this.state.logoMarginTop
           }]} />
        </View>
      )
    }
}
```

此示例通过更改边距来实现图像位置的动画效果。

Chapter 19: Animation API

Section 19.1: Animate an Image

```
class AnimatedImage extends Component {
    constructor(props){
        super(props)
        this.state = {
            logoMarginTop: new Animated.Value(200)
        }
    }
    componentDidMount(){
        Animated.timing(
            this.state.logoMarginTop,
            { toValue: 100 }
        ).start()
    }
    render () {
      return (
        <View>
           <Animated.Image source={require('../images/Logo.png')} style={[baseStyles.logo, {
             marginTop: this.state.logoMarginTop
           }]} />
        </View>
      )
    }
}
```

This example is animating the image position by changing the margin.

第20章：Android - 硬件返回按钮

第20.1节：检测Android中的硬件返回按钮按下事件

```
BackAndroid.addEventListener('hardwareBackPress', function() {
    if (!this.onMainScreen()) {
        this.goBack();
        return true;
    }
    return false;
});
```

注意：this.onMainScreen() 和 this.goBack() 不是内置函数，你还需要实现它们。
(https://github.com/immidi/react-native/commit/ed7e0fb31d842c63e8b8dc77ce795fac86e0f712)

第20.2节：BackAndroid 与 Navigator 的示例

这是一个如何将 React Native 的 BackAndroid 与 Navigator 一起使用的示例。

componentWillMount 注册了一个事件监听器来处理返回按钮的点击。它会检查历史堆栈中是否有另一个视图，如果有，则返回上一页——否则保持默认行为。

关于 BackAndroid 文档 和 Navigator 文档 的更多信息。

```
import React, { Component } from 'react'; // eslint-disable-line no-unused-vars

import {
BackAndroid,
Navigator,
} from 'react-native';

import SceneContainer from './Navigation/SceneContainer';
import RouteMapper from './Navigation/RouteMapper';

export default class AppContainer extends Component {

  constructor(props) {
    super(props);

    this.navigator;
  }

componentWillMount() {
BackAndroid.addEventListener('hardwareBackPress', () => {
      if (this.navigator && this.navigator.getCurrentRoutes().length > 1) {
        this.navigator.pop();
        return true;
      }
      return false;
    });
  }

renderScene(route, navigator) {
    this.navigator = navigator;

    return (
```

Chapter 20: Android - Hardware Back Button

Section 20.1: Detect Hardware back button presses in Android

```
BackAndroid.addEventListener('hardwareBackPress', function() {
    if (!this.onMainScreen()) {
        this.goBack();
        return true;
    }
    return false;
});
```

Note: `this.onMainScreen()` and `this.goBack()` are not built in functions, you also need to implement those.
(https://github.com/immidi/react-native/commit/ed7e0fb31d842c63e8b8dc77ce795fac86e0f712)

Section 20.2: Example of BackAndroid along with Navigator

This is an example on how to use React Native's `BackAndroid` along with the `Navigator`.

`componentWillMount` registers an event listener to handle the taps on the back button. It checks if there is another view in the history stack, and if there is one, it goes back -otherwise it keeps the default behaviour.

More information on the `BackAndroid docs` and the `Navigator docs`.

```
import React, { Component } from 'react'; // eslint-disable-line no-unused-vars

import {
  BackAndroid,
  Navigator,
} from 'react-native';

import SceneContainer from './Navigation/SceneContainer';
import RouteMapper from './Navigation/RouteMapper';

export default class AppContainer extends Component {

  constructor(props) {
    super(props);

    this.navigator;
  }

  componentWillMount() {
    BackAndroid.addEventListener('hardwareBackPress', () => {
      if (this.navigator && this.navigator.getCurrentRoutes().length > 1) {
        this.navigator.pop();
        return true;
      }
      return false;
    });
  }

  renderScene(route, navigator) {
    this.navigator = navigator;

    return (
      <SceneContainer
        title={route.title}
        route={route}
navigator={navigator}
        onBack={() => {
          if (route.index > 0) {
            navigator.pop();
          }
        }}
        {...this.props} />
    );
  }

render() {
    return (
      <Navigator
initialRoute={<View />}
        renderScene={this.renderScene.bind(this)}
        navigationBar={
          <Navigator.NavigationBar
            style={{backgroundColor: 'gray'}}
            routeMapper={RouteMapper} />
        } />
    );
  }
};
```

第20.3节：使用BackHandler和导航属性处理硬件返回按钮（不使用已弃用的BackAndroid和Navigator）

此示例将向您展示大多数流程中通常预期的返回导航。您需要根据预期行为将以下代码添加到每个屏幕。有两种情况：

1. 如果堆栈中有多个屏幕，设备返回按钮将显示上一个屏幕。
2. 如果堆栈中只有一个屏幕，设备返回按钮将退出应用程序。

案例 1：显示上一个屏幕

```
import { BackHandler } from 'react-native';

constructor(props) {
    super(props)
    this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
}

componentWillMount() {
BackHandler.addEventListener('hardwareBackPress', this.handleBackButtonClick);
}

componentWillUnmount() {
BackHandler.removeEventListener('hardwareBackPress', this.handleBackButtonClick);
}

handleBackButtonClick() {
    this.props.navigation.goBack(null);
    return true;
}
```

```
      <SceneContainer
        title={route.title}
        route={route}
        navigator={navigator}
        onBack={() => {
          if (route.index > 0) {
            navigator.pop();
          }
        }}
        {...this.props} />
    );
  }

  render() {
    return (
      <Navigator
        initialRoute={<View />}
        renderScene={this.renderScene.bind(this)}
        navigationBar={
          <Navigator.NavigationBar
            style={{backgroundColor: 'gray'}}
            routeMapper={RouteMapper} />
        } />
    );
  }
};
```

Section 20.3: Hardware back button handling using BackHandler and Navigation Properties (without using deprecated BackAndroid & deprecated Navigator)

This example will show you back navigation which is expected generally in most of the flows. You will have to add following code to every screen depending on expected behavior. There are 2 cases:

1. If there are more than 1 screen on stack, device back button will show previous screen.
2. If there is only 1 screen on stack, device back button will exit app.

Case 1: Show previous screen

```
import { BackHandler } from 'react-native';

constructor(props) {
    super(props)
    this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
}

componentWillMount() {
    BackHandler.addEventListener('hardwareBackPress', this.handleBackButtonClick);
}

componentWillUnmount() {
    BackHandler.removeEventListener('hardwareBackPress', this.handleBackButtonClick);
}

handleBackButtonClick() {
    this.props.navigation.goBack(null);
    return true;
}
```

重要提示：不要忘记在构造函数中绑定方法，并在 componentWillUnmount 中移除监听器。

案例 2：退出应用

在这种情况下，无需处理您想要退出应用的那个屏幕上的任何操作。

重要提示： 这应该是堆栈中唯一的屏幕。

第20.4节：使用BackHandler检测硬件返回按钮的示例

由于BackAndroid已被弃用，请使用BackHandler代替BackAndroid。

```
import { BackHandler } from 'react-native';

{...}
ComponentWillMount(){
BackHandler.addEventListener('hardwareBackPress',()=>{
      if (!this.onMainScreen()) {
        this.goBack();
        return true;
      }
      return false;
    });
  }
```

Important: Don't forget to bind method in constructor and to remove listener in componentWillUnmount.

Case 2: Exit App

In this case, no need to handle anything on that screen where you want to exit app.

Important: This should be only screen on stack.

Section 20.4: Example of Hardware back button detection using BackHandler

Since BackAndroid is deprecated. Use BackHandler instead of BackAndroid.

```
import { BackHandler } from 'react-native';

{...}
  ComponentWillMount(){
    BackHandler.addEventListener('hardwareBackPress',()=>{
      if (!this.onMainScreen()) {
        this.goBack();
        return true;
      }
      return false;
    });
  }
```

第21章：在设备上运行应用（Android版本）

第21.1节：在安卓设备上运行应用

1.adb devices
   - 你的手机是否显示在列表中？如果没有，请在手机上启用开发者模式，并通过USB连接。
2. adb reverse tcp:8081 tcp:8081 ：
   - 为了正确连接你的手机，并让React-Native在构建时识别它。（**注意：安卓版本5及以上。**）
3. react-native run-android ：
   - 在手机上运行应用。
4. react-native start ：
   - 用于启动本地开发服务器（必需）。如果你使用的是最新版React-native，该服务器会自动启动。

Chapter 21: Run an app on device (Android Version)

Section 21.1: Running an app on Android Device

1. `adb devices`
   - Is your phone displaying? If not, enable developer mode on your phone, and connect it by USB.
2. `adb reverse tcp:8081 tcp:8081`:
   - In order to link correctly your phone and that React-Native recognize him during build. (**NOTE:`Android Version` 5 or above.**)
3. `react-native run-android`:
   - To run the app on your phone.
4. `react-native start`:
   - In order to start a local server for development (mandatory). This server is automatically started if you use the last version of React-native.

第22章：原生模块

第22.1节：创建你的原生模块（iOS）

介绍

来自 http://facebook.github.io/react-native/docs/native-modules-ios.html

> 有时应用需要访问平台API，而React Native尚未提供相应的模块。也许你想重用一些现有的Objective-C、Swift或C++代码，而不必用JavaScript重新实现，或者编写一些高性能、多线程的代码，比如图像处理、数据库，或其他各种高级扩展。

原生模块只是一个实现了RCTBridgeModule协议的Objective-C类。

示例

在你的Xcode项目中创建一个新文件，选择Cocoa Touch Class，在创建向导中为你的类命名（例如NativeModule），将其设置为NSObject的子类，语言选择Objective-C。

这将创建两个文件NativeModuleEx.h和NativeModuleEx.m

你需要在NativeModuleEx.h文件中导入RCTBridgeModule.h，内容如下：

```
#import <Foundation/Foundation.h>
#import "RCTBridgeModule.h"

@interface NativeModuleEx : NSObject <RCTBridgeModule>

@end
```

在你的NativeModuleEx.m中添加以下代码：

```
#import "NativeModuleEx.h"

@implementation NativeModuleEx

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(testModule:(NSString *)string )
{
NSLog(@"字符串 '%@' 来自 JavaScript！", string);
}

@end
```

RCT_EXPORT_MODULE() 将使你的模块在 JavaScript 中可访问，你可以传入一个可选参数来指定它的名称。如果没有提供名称，则默认使用 Objective-C 类名。

RCT_EXPORT_METHOD() 将把你的方法暴露给 JavaScript，只有使用此宏导出的方法才能在 JavaScript 中访问。

最后，在你的 JavaScript 中可以按如下方式调用你的方法：

Chapter 22: Native Modules

Section 22.1: Create your Native Module (IOS)

Introduction

from http://facebook.github.io/react-native/docs/native-modules-ios.html

> Sometimes an app needs access to platform API, and React Native doesn't have a corresponding module yet. Maybe you want to reuse some existing Objective-C, Swift or C++ code without having to reimplement it in JavaScript, or write some high performance, multi-threaded code such as for image processing, a database, or any number of advanced extensions.

A Native Module is simply an Objective-C Class that implements the `RCTBridgeModule` protocol.

Example

In your Xcode project create a new file and select **Cocoa Touch Class**, in the creation wizard choose a name for your Class (*e.g. NativeModule*), make it a **Subclass of**: `NSObject` and choose `Objective-C` for the language.

This will create two files `NativeModuleEx.h` and `NativeModuleEx.m`

You will need to import `RCTBridgeModule.h` to your `NativeModuleEx.h` file as it follows:

```
#import <Foundation/Foundation.h>
#import "RCTBridgeModule.h"

@interface NativeModuleEx : NSObject <RCTBridgeModule>

@end
```

In your `NativeModuleEx.m` add the following code:

```
#import "NativeModuleEx.h"

@implementation NativeModuleEx

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(testModule:(NSString *)string )
{
  NSLog(@"The string '%@' comes from JavaScript! ", string);
}

@end
```

`RCT_EXPORT_MODULE()` will make your module accessible in JavaScript, you can pass it an optional argument to specify its name. If no name is provided it will match the Objective-C class name.

`RCT_EXPORT_METHOD()` will expose your method to JavaScript, only the methods you export using this macro will be accessible in JavaScript.

Finally, in your JavaScript you can call your method as it follows:

```
import { NativeModules } from 'react-native';

var NativeModuleEx = NativeModules.NativeModuleEx;

NativeModuleEx.testModule('某个字符串！');
```

```
import { NativeModules } from 'react-native';

var NativeModuleEx = NativeModules.NativeModuleEx;

NativeModuleEx.testModule('Some String !');
```

第23章：链接本地API

链接API使您能够在应用程序之间发送和接收链接。例如，打开电话应用并拨打号码，或打开谷歌地图并开始导航到选定的目的地。您还可以利用链接功能，使您的应用能够响应来自其他应用打开的链接。

要使用Linking，您需要先从react-native中导入它

```
import {Linking} from 'react-native'
```

第23.1节：发出链接

要打开链接，请调用openURL。

```
Linking.openURL(url)
.catch(err => console.error('发生错误 ', err))
```

首选方法是事先检查是否有已安装的应用可以处理给定的URL。

```
Linking.canOpenURL(url)
.then(supported => {
  if (!supported) {
console.log('不支持的URL: ' + url)
  } else {
    return Linking.openURL(url)
  }
}).catch(err => console.error('发生错误 ', err))
```

URI 方案

目标应用	示例	参考
网页浏览器	`https://stackoverflow.com`	
电话	tel:1-408-555-5555	苹果
邮件	mailto:email@example.com	苹果
短信	sms:1-408-555-1212	苹果
苹果地图	`http://maps.apple.com/?ll=37.484847,-122.148386`	苹果
谷歌地图	`geo:37.7749,-122.4194`	谷歌
iTunes	查看 iTunes 链接生成器	苹果
脸书	fb://profile	Stack Overflow
YouTube	http://www.youtube.com/v/oHg5SJYRHA0	苹果
Facetime	facetime://user@example.com	苹果
iOS 日历	`calshow:514300000` [1]	iPhoneDevWiki

[1] 在自 2001 年 1 月 1 日（UTC？）起的指定秒数处打开日历。出于某种原因，苹果未对该 API 进行文档说明。

第 23.2 节：传入链接

您可以检测应用是否由外部 URL 启动。

```
componentDidMount() {
  const url = Linking.getInitialURL()
  .then((url) => {
    if (url) {
console.log('初始 URL 是: ' + url)
```

Chapter 23: Linking Native API

Linking API enables you to both send and receive links between applications. For example, opening the Phone app with number dialed in or opening the Google Maps and starting a navigation to a chosen destination. You can also utilise Linking to make your app able to respond to links opening it from other applications.

To use `Linking` you need to first import it from `react-native`

```
import {Linking} from 'react-native'
```

Section 23.1: Outgoing Links

To open a link call openURL.

```
Linking.openURL(url)
.catch(err => console.error('An error occurred ', err))
```

The preferred method is to check if any installed app can handle a given URL beforehand.

```
Linking.canOpenURL(url)
.then(supported => {
  if (!supported) {
    console.log('Unsupported URL: ' + url)
  } else {
    return Linking.openURL(url)
  }
}).catch(err => console.error('An error occurred ', err))
```

URI Schemes

Target App	Example	Reference
Web Browser	`https://stackoverflow.com`	
Phone	`tel:1-408-555-5555`	Apple
Mail	`mailto:email@example.com`	Apple
SMS	`sms:1-408-555-1212`	Apple
Apple Maps	`http://maps.apple.com/?ll=37.484847,-122.148386`	Apple
Google Maps	`geo:37.7749,-122.4194`	Google
iTunes	See iTunes Link Maker	Apple
Facebook	`fb://profile`	Stack Overflow
YouTube	`http://www.youtube.com/v/oHg5SJYRHA0`	Apple
Facetime	`facetime://user@example.com`	Apple
iOS Calendar	`calshow:514300000` [1]	iPhoneDevWiki

[1] Opens the calendar at the stated number of seconds since 1. 1. 2001 (UTC?). For some reason this API is undocumented by Apple.

Section 23.2: Incomming Links

You can detect when your app is launched from an external URL.

```
componentDidMount() {
  const url = Linking.getInitialURL()
  .then((url) => {
    if (url) {
      console.log('Initial url is: ' + url)
    }
  }).catch(err => console.error('发生错误 ', err))
}
```

要在 iOS 上启用此功能，请将 Link RCTLinking 添加到您的项目中。

要在 Android 上启用此功能，请按照以下步骤操作。

```
    }
  }).catch(err => console.error('An error occurred ', err))
}
```

To enable this on iOS Link RCTLinking to your project.

To enable this on Android, follow these steps.

第24章：React Native中的ESLint

这是关于react-native中ESLint规则说明的主题。

第24.1节：如何开始

强烈建议在您的react-native项目中使用ESLint。ESLint是一个使用社区提供的特定规则进行代码验证的工具。

对于react-native，您可以使用javascript、react和react-native的规则集。

您可以在这里找到带有动机和解释的常用javascript ESLint规则：https://github.com/eslint/eslint/tree/master/docs/rules 。您可以通过在您的 .eslintr.json 文件的 'extends' 节点中添加 'eslint:recommended' 来简单地添加ESLint开发者提供的现成规则集。（"extends": ["eslint:recommended"]）更多关于ESLint配置的信息您可以阅读这里：http://eslint.org/docs/developer-guide/development-environment 。强烈建议阅读这款极其有用工具的完整文档。

接下来，您可以在这里找到关于ESLint react插件规则的完整文档：https://github.com/yannickcr/eslint-plugin-react/tree/master/docs/rules 。重要提示：并非所有来自 react 的规则都与 react-native 相关。例如：react/display-name 和 react/no-unknown-property。另一些规则是每个 react-native 项目"必须有"的，比如 react/jsx-no-bind 和 react/jsx-key。

选择自己的规则集时要非常小心。

最后，有一个专门针对 react-native 的插件：https://github.com/intellicode/eslint-plugin-react-native 注意：如果你将样式拆分到单独的文件，规则 react-native/no-inline-styles 将无法生效。

为了使该工具在 react-native 环境中正常工作，你可能需要在配置中将 'env' 的值设置为：

```
"env": {
"browser": true,
"es6": true,
"amd": true
},
```

ESLint 是开发高质量产品的关键工具。

Chapter 24: ESLint in React Native

This is the topic for ESLint rules explanation for react-native.

Section 24.1: How to start

It's highly recommended to use ESLint in your project on react-native. ESLint is a tool for code validation using specific rules provided by community.

For react-native you can use rulesets for javascript, react and react-native.

Common ESLint rules with motivation and explanations for javascript you can find here: https://github.com/eslint/eslint/tree/master/docs/rules . You can simply add ready ruleset from ESLint developers by adding in your .eslintr.json to 'extends' node 'eslint:recommended'. ("extends": ["eslint:recommended"]) More about ESLint configuring you can read here: http://eslint.org/docs/developer-guide/development-environment . It's recommended to read full doc about this extremely useful tool.

Next, full docs about rules for ES Lint react plugin you can find here: https://github.com/yannickcr/eslint-plugin-react/tree/master/docs/rules . Important note: not all rules from react are relative to react-native. For example: react/display-name and react/no-unknown-property for example. Another rules are 'must have' for every project on react-native, such as react/jsx-no-bind and react/jsx-key.

Be very careful with choosing your own ruleset.

And finaly, there is a plugin explicidly for react-native: https://github.com/intellicode/eslint-plugin-react-native Note: If you split your styles in separate file, rule react-native/no-inline-styles will not work.

For correct working of this tool in react-native env you might need to set value or 'env' in your config to this:

```
"env": {
"browser": true,
"es6": true,
"amd": true
},
```

ESLint is a key tool for development of high quality product.

第25章：与 Firebase 的身份验证集成

```
//用你的应用 API 值替换 firebase 的值
import firebase from 'firebase';

componentWillMount() {

firebase.initializeApp({
        apiKey: "yourAPIKey",
        authDomain: "authDomainNAme",
        databaseURL: "yourDomainBaseURL",
        projectId: "yourProjectID",
storageBucket: "storageBUcketValue",
        messagingSenderId: "senderIdValue"
    });

firebase.auth().signInWithEmailAndPassword(email, password)
    .then(this.onLoginSuccess)
})
}
```

第25.1节：在React Native中使用Firebase进行身份验证

将firebase的值替换为您的应用API值：

```
import firebase from 'firebase';
componentWillMount() {
firebase.initializeApp({
  apiKey: "yourAPIKey",
  authDomain: "authDomainNAme",
  databaseURL: "yourDomainBaseURL",
  projectId: "yourProjectID",
storageBucket: "storageBUcketValue",
  messagingSenderId: "senderIdValue"
});
firebase.auth().signInWithEmailAndPassword(email, password)
  .then(this.onLoginSuccess)
.catch(() => {
firebase.auth().createUserWithEmailAndPassword(email, password)
      .then(this.onLoginSuccess)
.catch(this.onLoginFail)
  })
}
```

第25.2节：React Native - 使用Firebase的ListView

当我使用Firebase并想使用ListView时，我会这样做。

使用父组件从Firebase获取数据（Posts.js）：

Posts.js

```
import PostsList from './PostsList';

class Posts extends Component{
```

Chapter 25: Integration with Firebase for Authentication

```
//Replace firebase values with your app API values
import firebase from 'firebase';

componentWillMount() {

    firebase.initializeApp({
        apiKey: "yourAPIKey",
        authDomain: "authDomainNAme",
        databaseURL: "yourDomainBaseURL",
        projectId: "yourProjectID",
        storageBucket: "storageBUcketValue",
        messagingSenderId: "senderIdValue"
    });

    firebase.auth().signInWithEmailAndPassword(email, password)
    .then(this.onLoginSuccess)
})
}
```

Section 25.1: Authentication In React Native Using Firebase

Replace firebase values with your app api values:

```
import firebase from 'firebase';
componentWillMount() {
firebase.initializeApp({
  apiKey: "yourAPIKey",
  authDomain: "authDomainNAme",
  databaseURL: "yourDomainBaseURL",
  projectId: "yourProjectID",
  storageBucket: "storageBUcketValue",
  messagingSenderId: "senderIdValue"
});
    firebase.auth().signInWithEmailAndPassword(email, password)
  .then(this.onLoginSuccess)
  .catch(() => {
    firebase.auth().createUserWithEmailAndPassword(email, password)
      .then(this.onLoginSuccess)
      .catch(this.onLoginFail)
  })
}
```

Section 25.2: React Native - ListView with Firebase

This is what I do when I'm working with Firebase and I want to use ListView.

Use a parent component to retrieve the data from Firebase (Posts.js):

Posts.js

```
import PostsList from './PostsList';

class Posts extends Component{
```

```
constructor(props) {
        super(props);
        this.state = {
            posts: []
        }
    }

componentWillMount() {
firebase.database().ref('Posts/').on('value', function(data) {
            this.setState({ posts: data.val() });
        });
    }

render() {
        return <PostsList posts={this.state.posts}/>
    }
}
```

PostsList.js

```
class PostsList extends Component {
    constructor(props) {
        super(props);
        this.state = {
dataSource: new ListView.DataSource({
                rowHasChanged: (row1, row2) => row1 !== row2
            }),
        }
    }

getDataSource(posts: Array<any>): ListView.DataSource {
        if(!posts) return;
        return this.state.dataSource.cloneWithRows(posts);
    }

componentDidMount() {
        this.setState({dataSource: this.getDataSource(this.props.posts)});
    }

componentWillReceiveProps(props) {
        this.setState({dataSource: this.getDataSource(props.posts)});
    }

renderRow = (post) => {
        return (
            <View>
                <Text>{post.title}</Text>
                <Text>{post.content}</Text>
            </View>
        );
    }

render() {
        return(
            <ListView
dataSource={this.state.dataSource}
                renderRow={this.renderRow}
enableEmptySections={true}
            />
        );
    }
```

```
    constructor(props) {
        super(props);
        this.state = {
            posts: []
        }
    }

    componentWillMount() {
        firebase.database().ref('Posts/').on('value', function(data) {
            this.setState({ posts: data.val() });
        });
    }

    render() {
        return <PostsList posts={this.state.posts}/>
    }
}
```

PostsList.js

```
class PostsList extends Component {
    constructor(props) {
        super(props);
        this.state = {
            dataSource: new ListView.DataSource({
                rowHasChanged: (row1, row2) => row1 !== row2
            }),
        }
    }

    getDataSource(posts: Array<any>): ListView.DataSource {
        if(!posts) return;
        return this.state.dataSource.cloneWithRows(posts);
    }

    componentDidMount() {
        this.setState({dataSource: this.getDataSource(this.props.posts)});
    }

    componentWillReceiveProps(props) {
        this.setState({dataSource: this.getDataSource(props.posts)});
    }

    renderRow = (post) => {
        return (
            <View>
                <Text>{post.title}</Text>
                <Text>{post.content}</Text>
            </View>
        );
    }

    render() {
        return(
            <ListView
                dataSource={this.state.dataSource}
                renderRow={this.renderRow}
                enableEmptySections={true}
            />
        );
    }
}
```

我想指出，在Posts.js中，我没有导入firebase，因为你只需要在项目的主组件（有导航器的地方）导入一次，然后可以在任何地方使用它。

这是有人在我遇到ListView问题时建议的解决方案。我觉得分享出来会很好。

来源：[http://stackoverflow.com/questions/38414289/react-native-listview-not-rendering-data-from-firebase][1]

```
}
```

I want to point out that in `Posts.js`, I'm not importing `firebase` because you only need to import it once, in the main component of your project (where you have the navigator) and use it anywhere.

This is the solution someone suggested in a question I asked when I was struggling with ListView. I thought it would be nice to share it.

Source: [http://stackoverflow.com/questions/38414289/react-native-listview-not-rendering-data-from-firebase][1]

第26章：导航器最佳实践

第26.1节：导航器

导航器是React Native的默认导航器。一个导航器组件管理一个路由对象栈，并提供管理该栈的方法。

```
<Navigator
 ref={(navigator) => { this.navigator = navigator }}
  initialRoute={{ id: 'route1', title: 'Route 1' }}
  renderScene={this.renderScene.bind(this)}
configureScene={(route) => Navigator.SceneConfigs.FloatFromRight}
  style={{ flex: 1 }}
  navigationBar={
// 见下文"管理导航栏"
    <Navigator.NavigationBar routeMapper={this.routeMapper} />
  }
/>
```

管理路由栈

首先，注意initialRoute属性。路由只是一个JavaScript对象，可以具有你想要的任何结构和任何值。它是你在导航堆栈中传递值和方法的主要方式。

Navigator根据其renderScene属性返回的值来决定渲染内容。

```
renderScene(route, navigator) {
  if (route.id === 'route1') {
    return <ExampleScene navigator={navigator} title={route.title} />; // 见下文
  } else if (route.id === 'route2') {
    return <ExampleScene navigator={navigator} title={route.title} />; // 见下文
  }
}
```

让我们设想一下这个例子中ExampleScene的实现：

```
function ExampleScene(props) {

  function forward() {
    // 这个路由对象将传递给我们上面定义的`renderScene`函数。
props.navigator.push({ id: 'route2', title: 'Route 2' });
  }

  function back() {
    // `pop` 只是从 `Navigator` 的堆栈中弹出一个路由对象
     props.navigator.pop();
  }

  return (
    <View>
      <Text>{props.title}</Text>
      <TouchableOpacity onPress={forward}>
        <Text>前进!</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={back}>
        <Text>后退!</Text>
      </TouchableOpacity>
```

Chapter 26: Navigator Best Practices

Section 26.1: Navigator

`Navigator` is React Native's default navigator. A `Navigator` component manages a *stack* of route objects, and provides methods for managing that stack.

```
<Navigator
  ref={(navigator) => { this.navigator = navigator }}
  initialRoute={{ id: 'route1', title: 'Route 1' }}
  renderScene={this.renderScene.bind(this)}
  configureScene={(route) => Navigator.SceneConfigs.FloatFromRight}
  style={{ flex: 1 }}
  navigationBar={
    // see "Managing the Navigation Bar" below
    <Navigator.NavigationBar routeMapper={this.routeMapper} />
  }
/>
```

Managing the Route Stack

First of all, notice the `initialRoute` prop. A route is simply a javascript object, and can take whatever shape you want, and have whatever values you want. It's the primary way you'll pass values and methods between components in your navigation stack.

The `Navigator` knows what to render based on the value returned from its `renderScene` prop.

```
renderScene(route, navigator) {
  if (route.id === 'route1') {
    return <ExampleScene navigator={navigator} title={route.title} />; // see below
  } else if (route.id === 'route2') {
    return <ExampleScene navigator={navigator} title={route.title} />; // see below
  }
}
```

Let's imagine an implementation of `ExampleScene` in this example:

```
function ExampleScene(props) {

  function forward() {
    // this route object will passed along to our `renderScene` function we defined above.
    props.navigator.push({ id: 'route2', title: 'Route 2' });
  }

  function back() {
    // `pop` simply pops one route object off the `Navigator`'s stack
    props.navigator.pop();
  }

  return (
    <View>
      <Text>{props.title}</Text>
      <TouchableOpacity onPress={forward}>
        <Text>Go forward!</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={back}>
        <Text>Go Back!</Text>
      </TouchableOpacity>
    </View>
  );
}
```

配置导航器

你可以通过 configureScene 属性配置 Navigator 的过渡效果。该属性是一个函数，接收
route 对象，并需要返回一个配置对象。以下是可用的配置对象：

- Navigator.SceneConfigs.PushFromRight（默认）
- Navigator.SceneConfigs.FloatFromRight
- Navigator.SceneConfigs.FloatFromLeft
- Navigator.SceneConfigs.FloatFromBottom
- Navigator.SceneConfigs.FloatFromBottomAndroid
- Navigator.SceneConfigs.FadeAndroid
- Navigator.SceneConfigs.HorizontalSwipeJump
- Navigator.SceneConfigs.HorizontalSwipeJumpFromRight
- Navigator.SceneConfigs.VerticalUpSwipeJump
- Navigator.SceneConfigs.VerticalDownSwipeJump

您可以返回这些对象中的任意一个而不做修改，或者您可以修改配置对象以自定义导航过渡效果。例如，修改边缘触发宽度以更接近模拟iOS
UINavigationController的interactivePopGestureRecognizer：

```
configureScene={(route) => {
  return {
...Navigator.SceneConfigs.FloatFromRight,
    gestures: {
pop: {
...Navigator.SceneConfigs.FloatFromRight.gestures.pop,
        edgeHitWidth: Dimensions.get('window').width / 2,
      },
    },
  };
}}
```

管理导航栏

Navigator组件带有一个navigationBar属性，理论上可以接受任何配置正确的
React组件。但最常见的实现是使用默认的Navigator.NavigationBar。它接受一个
routeMapper属性，您可以用它根据路由配置导航栏的外观。

routeMapper 是一个常规的 JavaScript 对象，包含三个函数：Title、RightButton 和 LeftButton。例如：

```
const routeMapper = {

LeftButton(route, navigator, index, navState) {
    if (index === 0) {
      return null;
    }

    return (
      <TouchableOpacity
onPress={() => navigator.pop()}
        style={styles.navBarLeftButton}
      >
```

```
    </View>
  );
}
```

Configuring the Navigator

You can configure the Navigator's transitions with the configureScene prop. This is a function that's passed the route object, and needs to return a configuration object. These are the available configuration objects:

- Navigator.SceneConfigs.PushFromRight (default)
- Navigator.SceneConfigs.FloatFromRight
- Navigator.SceneConfigs.FloatFromLeft
- Navigator.SceneConfigs.FloatFromBottom
- Navigator.SceneConfigs.FloatFromBottomAndroid
- Navigator.SceneConfigs.FadeAndroid
- Navigator.SceneConfigs.HorizontalSwipeJump
- Navigator.SceneConfigs.HorizontalSwipeJumpFromRight
- Navigator.SceneConfigs.VerticalUpSwipeJump
- Navigator.SceneConfigs.VerticalDownSwipeJump

You can return one of these objects without modification, or you can modify the configuration object to customize the navigation transitions. For example, to modify the edge hit width to more closely emulate the iOS UINavigationController's interactivePopGestureRecognizer:

```
configureScene={(route) => {
  return {
    ...Navigator.SceneConfigs.FloatFromRight,
    gestures: {
      pop: {
        ...Navigator.SceneConfigs.FloatFromRight.gestures.pop,
        edgeHitWidth: Dimensions.get('window').width / 2,
      },
    },
  };
}}
```

Managing the NavigationBar

The Navigator component comes with a navigationBar prop, which can theoretically take any properly configured React component. But the most common implementation uses the default Navigator.NavigationBar. This takes a routeMapper prop that you can use to configure the appearance of the navigation bar based on the route.

A routeMapper is a regular javascript object with three functions: Title, RightButton, and LeftButton. For example:

```
const routeMapper = {

  LeftButton(route, navigator, index, navState) {
    if (index === 0) {
      return null;
    }

    return (
      <TouchableOpacity
        onPress={() => navigator.pop()}
        style={styles.navBarLeftButton}
      >
```

```
        <Text>返回</Text>
      </TouchableOpacity>
    );
  },

RightButton(route, navigator, index, navState) {
    return (
      <TouchableOpacity
onPress={route.handleRightButtonClick}
        style={styles.navBarRightButton}
      >
        <Text>下一步</Text>
      </TouchableOpacity>
   );
  },

Title(路由,导航器,索引,导航状态) {
    return (
      <文本>
        {路由.标题}
      </文本>
    );
  },
};
```

查看更多

有关每个属性的更详细文档，请参见官方 React Native 文档中关于Navigator的部分，以及 React Native 关于使用导航器的指南。

第26.2节：在 React Native 应用中使用 react-navigation 进行导航

借助react-navigation，你可以非常轻松地为你的应用添加导航功能。

安装 react-navigation

```
npm install --save react-navigation
```

示例：

```
import { Button, View, Text, AppRegistry } from 'react-native';
import { StackNavigator } from 'react-navigation';

const App = StackNavigator({
  FirstPage: {screen: FirstPage},
  SecondPage: {screen: SecondPage},
});

class FirstPage extends React.Component {
  static navigationOptions = {
    title: '欢迎',
  };
render() {
    const { navigate } = this.props.navigation;

    return (
      <Button
title='前往第二页'
        onPress={() =>
```

```
        <Text>Back</Text>
      </TouchableOpacity>
    );
  },

  RightButton(route, navigator, index, navState) {
    return (
      <TouchableOpacity
        onPress={route.handleRightButtonClick}
        style={styles.navBarRightButton}
      >
        <Text>Next</Text>
      </TouchableOpacity>
   );
  },

  Title(route, navigator, index, navState) {
    return (
      <Text>
        {route.title}
      </Text>
    );
  },
};
```

See more

For more detailed documentation of each prop, see the the official React Native Documentation for Navigator, and the React Native guide on Using Navigators.

Section 26.2: Use react-navigation for navigation in react native apps

With the help of react-navigation, you can add navigation to your app really easy.

Install react-navigation

```
npm install --save react-navigation
```

Example:

```
import { Button, View, Text, AppRegistry } from 'react-native';
import { StackNavigator } from 'react-navigation';

const App = StackNavigator({
  FirstPage: {screen: FirstPage},
  SecondPage: {screen: SecondPage},
});

class FirstPage extends React.Component {
  static navigationOptions = {
    title: 'Welcome',
  };
  render() {
    const { navigate } = this.props.navigation;

    return (
      <Button
        title='Go to Second Page'
        onPress={() =>
navigate('SecondPage', { name: 'Awesomepankaj' })
          }
        />
      );
    }
}

class SecondPage extends React.Component {
  static navigationOptions = ({navigation}) => ({
    title: navigation.state.params.name,
  });

render() {
    const { goBack } = this.props.navigation;
    return (
      <View>
        <Text>欢迎来到第二页</Text>
        <Button
title="返回首页"
         onPress={() => goBack()}
         />
      </View>
    );
  }
}
```

第26.3节：使用 react-native-router-flux 的 react-native 导航

通过 npm `install --save` react-native-router-flux 安装

在 react-native-router-flux 中，每个路由称为一个 **<Scene >**

> <Scene key="home" component={LogIn} title="首页" initial />

key 用于引用特定场景的唯一字符串。

`component` 显示哪个组件，这里是

`title` 创建一个导航栏并给它一个标题"首页"

`initial` 这是应用的第一个屏幕吗

> 示例：

```
import React 来自 'react';
import { Scene, Router } 来自 'react-native-router-flux';
import LogIn 来自 './components/LogIn';
import SecondPage 来自 './components/SecondPage';

const RouterComponent = () => {
  return (
    <路由器>
      <场景键="login" 组件={登录} 标题="登录表单" 初始 />
      <场景键="secondPage" 组件={第二页} 标题="主页" />
    </路由器>
```

```
          navigate('SecondPage', { name: 'Awesomepankaj' })
        }
      />
    );
  }
}

class SecondPage extends React.Component {
  static navigationOptions = ({navigation}) => ({
    title: navigation.state.params.name,
  });

  render() {
    const { goBack } = this.props.navigation;
    return (
      <View>
        <Text>Welcome to Second Page</Text>
        <Button
          title="Go back to First Page"
          onPress={() => goBack()}
        />
      </View>
    );
  }
}
```

Section 26.3: react-native Navigation with react-native-router-flux

Install by using npm `install --save react-native-router-flux`

In react-native-router-flux, each route is called a **<Scene>**

> `<Scene key="home" component={LogIn} title="Home" initial />`

`key` A unique string that can be used to refer to the particular scene.

`component` Which component to show, here it's

`title` make a NavBar and give it a title 'Home'

`initial` Is this the first screen of the App

> Example:

```
import React from 'react';
import { Scene, Router } from 'react-native-router-flux';
import LogIn from './components/LogIn';
import SecondPage from './components/SecondPage';

const RouterComponent = () => {
  return (
    <Router>
      <Scene key="login" component={LogIn} title="Login Form" initial />
      <Scene key="secondPage" component={SecondPage} title="Home" />
    </Router>
  );
};

导出 默认 路由器组件;
```

在主 App.js（入口文件）中导入此文件并渲染。更多信息可访问此链接。

```
  );
};

export default RouterComponent;
```

Import this file in the main App.js(index file) and render it. For more information can visit this link.

第27章：带按钮的导航器从页面注入

第27.1节：介绍

与其让包含按钮导航器的主 js 文件臃肿，不如在任何需要的页面按需注入按钮，这样更简洁。

```
//在"主页"页面，我想让右侧导航按钮显示
//一个位于"主页"组件中的设置模态框。

componentWillMount() {
  this.props.route.navbarTitle = "主页";

  this.props.route.rightNavButton = {
    text: "设置",
onPress: this._显示设置模态框.bind(this)
  };
}
```

第27.2节：完整注释示例

```
'use strict';

import React, {Component} from 'react';
import ReactNative from 'react-native';

const {
AppRegistry,
  StyleSheet,
  Text,
View,
Navigator,
  Alert,
TouchableHighlight
} = ReactNative;


//这是包含导航器内容的应用容器
class AppContainer extends Component {

    renderScene(route, navigator) {
        switch(route.name) {
            case "Home":
      //你必须将路由作为属性传递，才能使此技巧正常工作
                  return <Home route={route} navigator={navigator} {...route.passProps}  />
            default:
            return (
        <Text route={route}
style={styles.container}>
您的路由名称可能不正确 {JSON.stringify(route)}
            </Text>
      );
        }
    }

render() {
    return (
```

Chapter 27: Navigator with buttons injected from pages

Section 27.1: Introduction

Instead of bloating your main js file that contains your navigator with buttons. It's cleaner to just inject buttons on-demand in any page that you need.

```
//In the page "Home", I want to have the right nav button to show
//a settings modal that resides in "Home" component.

componentWillMount() {
  this.props.route.navbarTitle = "Home";

  this.props.route.rightNavButton = {
    text: "Settings",
    onPress: this._ShowSettingsModal.bind(this)
  };
}
```

Section 27.2: Full commented example

```
'use strict';

import React, {Component} from 'react';
import ReactNative from 'react-native';

const {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Navigator,
  Alert,
  TouchableHighlight
} = ReactNative;


//This is the app container that contains the navigator stuff
class AppContainer extends Component {

    renderScene(route, navigator) {
        switch(route.name) {
            case "Home":
      //You must pass route as a prop for this trick to work properly
            return <Home route={route} navigator={navigator} {...route.passProps}  />
            default:
            return (
        <Text route={route}
        style={styles.container}>
            Your route name is probably incorrect {JSON.stringify(route)}
            </Text>
      );
        }
    }

  render() {
    return (
      <Navigator
navigationBar={
          <Navigator.NavigationBar
style={ styles.navbar }
            routeMapper={ NavigationBarRouteMapper } />
        }

initialRoute={{ name: 'Home' }}
        renderScene={ this.renderScene }

      />
    );
  }
}


//这里没什么特别的，除了检查注入的按钮。
//注意我们是如何检查路由对象中是否有注入的按钮。
//此外，当页面不是索引0（例如不是首页）时，我们会显示一个"返回"按钮。
var NavigationBarRouteMapper = {
  LeftButton(route, navigator, index, navState) {
    if(route.leftNavButton) {
      return (
        <TouchableHighlight
style={styles.leftNavButton}
underlayColor="transparent"
onPress={route.leftNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.leftNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
    else if(route.enableBackButton) {
      return (
        <TouchableHighlight
style={styles.leftNavButton}
       underlayColor="transparent"
       onPress={() => navigator.pop() }>
          <Text style={styles.navbarButtonText}>返回</Text>
        </TouchableHighlight>
      );
    }
  },
RightButton(route, navigator, index, navState) {
    if(route.rightNavButton) {
      return (
        <TouchableHighlight
style={styles.rightNavButton}
underlayColor="transparent"
onPress={route.rightNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.rightNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
  },
Title(路由,导航器,索引,导航状态) {
    //你也可以注入标题。如果不注入，我们将使用路由名称。
    return (<Text style={styles.navbarTitle}>{route.navbarTitle || route.name}</Text>);
  }
};

//这被视为导航器显示的子页面
class Home extends Component {
```

```
      <Navigator
        navigationBar={
          <Navigator.NavigationBar
            style={ styles.navbar }
            routeMapper={ NavigationBarRouteMapper } />
        }

        initialRoute={{ name: 'Home' }}
        renderScene={ this.renderScene }

      />
    );
  }
}


//Nothing fancy here, except for checking for injected buttons.
//Notice how we are checking if there are injected buttons inside the route object.
//Also, we are showing a "Back" button when the page is not at index-0 (e.g. not home)
var NavigationBarRouteMapper = {
  LeftButton(route, navigator, index, navState) {
    if(route.leftNavButton) {
      return (
        <TouchableHighlight
        style={styles.leftNavButton}
        underlayColor="transparent"
        onPress={route.leftNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.leftNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
    else if(route.enableBackButton) {
      return (
        <TouchableHighlight
        style={styles.leftNavButton}
        underlayColor="transparent"
        onPress={() => navigator.pop() }>
          <Text style={styles.navbarButtonText}>Back</Text>
        </TouchableHighlight>
      );
    }
  },
  RightButton(route, navigator, index, navState) {
    if(route.rightNavButton) {
      return (
        <TouchableHighlight
        style={styles.rightNavButton}
        underlayColor="transparent"
        onPress={route.rightNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.rightNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
  },
  Title(route, navigator, index, navState) {
    //You can inject the title aswell.  If you don't we'll use the route name.
    return (<Text style={styles.navbarTitle}>{route.navbarTitle || route.name}</Text>);
  }
};

//This is considered a sub-page that navigator is showing
class Home extends Component {
  //这个技巧依赖于 componentWillMount 在导航栏创建之前触发
componentWillMount() {
        this.props.route.navbarTitle = "首页";

        this.props.route.rightNavButton = {
            text: "按钮",
            onPress: this._doSomething.bind(this)
        };
    }

  //此方法将在按下注入的按钮时调用。
_doSomething() {
        Alert.alert(
        '太棒了，是吧？',
        null,
        [
          {text: '确实'},
        ]
      )
  }

render() {
    return (
      <View style={styles.container}>
            <Text>你在家</Text>
        </View>
      );
  }
}

var styles = StyleSheet.create({
  container: {
flex: 1,
justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
    marginTop: 66
  },
navbar: {
backgroundColor: '#ffffff',
  },
navbarTitle: {
    marginVertical: 10,
    fontSize: 17
  },
leftNavButton: {
    marginVertical: 10,
    paddingLeft: 8,
 },
rightNavButton: {
    marginVertical: 10,
    paddingRight: 8,
  },
navbarButtonText: {
    fontSize: 17,
    color: "#007AFF"
  }
});

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

```
  //This trick depends on that componentWillMount fires before the navbar is created
  componentWillMount() {
        this.props.route.navbarTitle = "Home";

        this.props.route.rightNavButton = {
            text: "Button",
            onPress: this._doSomething.bind(this)
        };
    }

  //This method will be invoked by pressing the injected button.
  _doSomething() {
      Alert.alert(
      'Awesome, eh?',
      null,
      [
        {text: 'Indeed'},
      ]
    )
  }

  render() {
    return (
      <View style={styles.container}>
            <Text>You are home</Text>
        </View>
    );
  }
}

var styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
    marginTop: 66
  },
  navbar: {
    backgroundColor: '#ffffff',
  },
  navbarTitle: {
    marginVertical: 10,
    fontSize: 17
  },
  leftNavButton: {
    marginVertical: 10,
    paddingLeft: 8,
 },
  rightNavButton: {
    marginVertical: 10,
    paddingRight: 8,
  },
  navbarButtonText: {
    fontSize: 17,
    color: "#007AFF"
  }
});

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

第28章：为安卓创建可分享的APK

创建APK（已签名和未签名）的步骤，您可以使用CLI安装到设备上并进行分享：

问题陈述：我已经构建了我的应用，我可以在本地模拟器上运行它（也可以通过更改调试服务器在我的安卓设备上运行）。但是，我想构建一个APK，可以发送给没有开发服务器访问权限的人，并且我希望他们能够测试该应用程序。

第28.1节：创建用于签名APK的密钥

```
keytool -genkey -v -keystore my-app-key.keystore -alias my-app-alias -keyalg RSA -keysize 2048 -
validity 10000
```

提示时请输入密码

第28.2节：一旦生成密钥，使用它来生成可安装的构建：

```
react-native bundle --platform android --dev false --entry-file index.android.js \
--bundle-output android/app/src/main/assets/index.android.bundle
--assets-dest android/app/src/main/res/
```

第28.3节：使用gradle生成构建

```
cd android && ./gradlew assembleRelease
```

第28.4节：上传或分享生成的APK

将APK上传到手机。-r标志将替换现有应用（如果存在）

```
adb install -r ./app/build/outputs/apk/app-release-unsigned.apk
```

可分享的签名APK位于：

```
./app/build/outputs/apk/app-release.apk
```

Chapter 28: Create a shareable APK for android

Steps to create an APK (signed and unsigned) which you can install on a device using CLI and share as well:

Problem statement: I've built my app, I can run it on my local emulator (and also on my android device by changing debug server). But, I want to build an apk that I can send to someone without access to development server and I want them to be able to test application.

Section 28.1: Create a key to sign the APK

```
keytool -genkey -v -keystore my-app-key.keystore -alias my-app-alias -keyalg RSA -keysize 2048 -
validity 10000
```

Use a password when prompted

Section 28.2: Once the key is generated, use it to generate the installable build:

```
react-native bundle --platform android --dev false --entry-file index.android.js \
--bundle-output android/app/src/main/assets/index.android.bundle \
--assets-dest android/app/src/main/res/
```

Section 28.3: Generate the build using gradle

```
cd android && ./gradlew assembleRelease
```

Section 28.4: Upload or share the generated APK

Upload the APK to your phone. The -r flag will replace the existing app (if it exists)

```
adb install -r ./app/build/outputs/apk/app-release-unsigned.apk
```

The shareable signed APK is located at:

```
./app/build/outputs/apk/app-release.apk
```

第29章：推送通知

我们可以通过使用npm模块**react-native-push-notification**（由 zo0r开发）为React Native应用添加推送通知功能。这支持跨平台开发。

安装

npm install --save react-native-push-notification

react-native link

第29.1节：推送通知简单设置

创建新项目 PushNotification

```
react-native init PushNotification
```

在 index.android.js 中写入以下内容

```
import React, { Component } from 'react';

import {
AppRegistry,
  StyleSheet,
  Text,
View,
按钮
} from 'react-native';

import PushNotification from 'react-native-push-notification';

export default class App extends Component {

constructor(props){
        super(props);

        this.NewNotification = this.NewNotification.bind(this);
      }

componentDidMount(){

        PushNotification.configure({

            // （必需）当远程或本地通知被打开或接收时调用
onNotification: function(notification) {
                console.log( 'NOTIFICATION:', notification );
            },

            // 是否应自动弹出初始通知
            // 默认值：true
popInitialNotification: true,

            /**
* （可选）默认值：true
* - 指定是否请求权限（iOS）和令牌（Android 和 iOS），
            * - 如果不请求，您必须稍后调用 PushNotificationsHandler.requestPermissions()
              */
requestPermissions: true,
```

Chapter 29: PushNotification

We can add Push Notification to react native app by using the npm module react-native-push-notification by zo0r. This enables for a cross platform development.

Installation

npm install --save react-native-push-notification

react-native link

Section 29.1: Push Notification Simple Setup

Create new project PushNotification

```
react-native init PushNotification
```

Put following in index.android.js

```
import React, { Component } from 'react';

import {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Button
} from 'react-native';

import PushNotification from 'react-native-push-notification';

export default class App extends Component {

    constructor(props){
        super(props);

        this.NewNotification = this.NewNotification.bind(this);
      }

    componentDidMount(){

        PushNotification.configure({

            // (required) Called when a remote or local notification is opened or received
            onNotification: function(notification) {
                console.log( 'NOTIFICATION:', notification );
            },

            // Should the initial notification be popped automatically
            // default: true
            popInitialNotification: true,

            /**
              * (optional) default: true
              * - Specified if permissions (ios) and token (android and ios) will requested or not,
              * - if not, you must call PushNotificationsHandler.requestPermissions() later
              */
            requestPermissions: true,
        });

    }

NewNotification(){

          let date = new Date(Date.now() + (this.state.seconds * 1000));

          //iOS 修复
        if(Platform.OS == "ios"){
            date = date.toISOString();
        }

PushNotification.localNotificationSchedule({
            message: "我的通知消息", // （必填）
            date: date,// （可选）用于设置延迟
largeIcon:""// 设为空以移除大图标//smallIcon: "ic_notification", // （可选）默认
             ："ic_notification"，回退为 "ic_launcher"

        });
    }

render() {

        return (
            <View style={styles.container}>
                <Text style={styles.welcome}>
               推送通知
                </Text>
                <View style={styles.Button} >
                <Button
onPress={()=>{this.NewNotification()}}
                  title="显示通知"
style={styles.Button}
                  color="#841584"
accessibilityLabel="显示通知"
                />
                </View>
            </View>
        );
      }
}

const styles = StyleSheet.create({
  container: {
flex: 1,
justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
Button:{
    margin: 10,
  }
});

AppRegistry.registerComponent('PushNotification', () => App);
```

```
        });

    }

      NewNotification(){

          let date = new Date(Date.now() + (this.state.seconds * 1000));

          //Fix for IOS
        if(Platform.OS == "ios"){
            date = date.toISOString();
        }

        PushNotification.localNotificationSchedule({
            message: "My Notification Message", // (required)
            date: date,// (optional) for setting delay
            largeIcon:""// set this blank for removing large icon
            //smallIcon: "ic_notification", // (optional) default: "ic_notification" with fallback
for "ic_launcher"
        });
    }

      render() {

        return (
            <View style={styles.container}>
                <Text style={styles.welcome}>
                  Push Notification
                </Text>
                <View style={styles.Button} >
                <Button
                  onPress={()=>{this.NewNotification()}}
                  title="Show Notification"
                  style={styles.Button}
                  color="#841584"
                  accessibilityLabel="Show Notification"
                />
                </View>
            </View>
        );
      }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
  Button:{
    margin: 10,
  }
});

AppRegistry.registerComponent('PushNotification', () => App);
```

第29.2节：从通知导航到场景

这里有一个简单的示例，演示如何根据通知跳转/打开特定的屏幕。
例如，当用户点击通知时，应用应打开并直接跳转到通知页面
而不是主页。

```
'use strict';

import React, { Component } from 'react' ;
import {
StyleSheet,
    Text,
查看,
Navigator,
TouchableOpacity,
AsyncStorage,
BackAndroid,
Platform,
} from 'react-native';
import PushNotification from 'react-native-push-notification';

let initialRoute = { id: 'loginview' }

export default class MainClass extends Component
{
constructor(props)
    {
        super(props);

        this.handleNotification = this.handleNotification.bind(this);
    }

handleNotification(notification)
    {
console.log('handleNotification');
        var notificationId = ''
        //你的逻辑，用于从通知中获取相关信息

    //这里你根据通知信息导航到应用中的某个场景
        this.navigator.push({ id: Constants.ITEM_VIEW_ID, item: item });
    }

componentDidMount()
    {
        var that = this;

PushNotification.configure({

            //（可选）当生成令牌（iOS 和 Android）时调用
onRegister: function(token) {
                console.log( 'TOKEN:', token );
            },

            // （必需）当远程或本地通知被打开或接收时调用
onNotification(notification) {
                console.log('onNotification')
                console.log( notification );

                that.handleNotification(notification);
            },
```

Section 29.2: Navigating to scene from Notification

Here's a simple example to demonstrate that how can we jump/open a specific screen based on the notification. For example, when a user clicks on the notification, the app should open and directly jump to notifications page instead of home page.

```
'use strict';

import React, { Component } from 'react';
import {
    StyleSheet,
    Text,
    View,
    Navigator,
    TouchableOpacity,
    AsyncStorage,
    BackAndroid,
    Platform,
} from 'react-native';
import PushNotification from 'react-native-push-notification';

let initialRoute = { id: 'loginview' }

export default class MainClass extends Component
{
    constructor(props)
    {
        super(props);

        this.handleNotification = this.handleNotification.bind(this);
    }

    handleNotification(notification)
    {
        console.log('handleNotification');
        var notificationId = ''
        //your logic to get relevant information from the notification

    //here you navigate to a scene in your app based on the notification info
        this.navigator.push({ id: Constants.ITEM_VIEW_ID, item: item });
    }

    componentDidMount()
    {
        var that = this;

        PushNotification.configure({

            // (optional) Called when Token is generated (iOS and Android)
            onRegister: function(token) {
                console.log( 'TOKEN:', token );
            },

            // (required) Called when a remote or local notification is opened or received
            onNotification(notification) {
                console.log('onNotification')
                console.log( notification );

                that.handleNotification(notification);
            },
            // 仅限安卓：（可选）GCM 发送者 ID。
senderID: "Vizido",

            // 仅限 iOS（可选）：默认值：all - 注册权限。
permissions: {
                alert: true,
                badge: true,
                sound: true
            },

            // 是否应自动弹出初始通知
            // 默认值：true
popInitialNotification: true,

            /**
* （可选）默认值：true
* - 指定是否请求权限（iOS）和令牌（Android 和 iOS），
            * - 如果不请求，您必须稍后调用 PushNotificationsHandler.requestPermissions()
              */
requestPermissions: true,
        });
    }

render()
    {

        return (
            <Navigator
ref={(nav) => this.navigator = nav }
                initialRoute={initialRoute}
renderScene={this.renderScene.bind(this)}
                configureScene={(route) =>
                    {
                        if (route.sceneConfig)
                        {
                            return route.sceneConfig;
                        }
                        return Navigator.SceneConfigs.FadeAndroid;
                    }
                }
            />
        );
    }

renderScene(route, navigator)
    {

        switch (route.id)
        {
            // 在这里进行路由处理
            case 'mainview':
                return ( <MainView navigator={navigator} /> );

            default:
                return ( <MainView navigator={navigator} /> );
        }
    }
}
```

```
            // ANDROID ONLY: (optional) GCM Sender ID.
            senderID: "Vizido",

            // IOS ONLY (optional): default: all - Permissions to register.
            permissions: {
                alert: true,
                badge: true,
                sound: true
            },

            // Should the initial notification be popped automatically
            // default: true
            popInitialNotification: true,

            /**
              * (optional) default: true
              * - Specified if permissions (ios) and token (android and ios) will requested or not,
              * - if not, you must call PushNotificationsHandler.requestPermissions() later
              */
            requestPermissions: true,
        });
    }

    render()
    {

        return (
            <Navigator
                ref={(nav) => this.navigator = nav }
                initialRoute={initialRoute}
                renderScene={this.renderScene.bind(this)}
                configureScene={(route) =>
                    {
                        if (route.sceneConfig)
                        {
                            return route.sceneConfig;
                        }
                        return Navigator.SceneConfigs.FadeAndroid;
                    }
                }
            />
        );
    }

    renderScene(route, navigator)
    {

        switch (route.id)
        {
            // do your routing here
            case 'mainview':
                return ( <MainView navigator={navigator} /> );

            default:
                return ( <MainView navigator={navigator} /> );
        }
    }
}
```

第30章：渲染最佳实践

关于特定组件.render方法行为的重要说明主题。

第30.1节：JSX中的函数

为了更好的性能，避免在JSX中使用数组（lambda）函数非常重要。

如在 https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md 中所解释：

> 在JSX属性中调用bind或使用箭头函数会在每次渲染时创建一个全新的函数。这对性能不利，因为这会导致垃圾回收器被调用的次数远远超过必要的次数。它还可能导致不必要的重新渲染，如果将一个全新的函数作为属性传递给一个使用引用相等检查来判断是否需要更新的组件。

所以如果有如下的jsx代码块：

```
                    <TextInput
                     onChangeValue={  value => this.handleValueChanging(value) }
                 />
```

或者

```
                    <button onClick={ this.handleClick.bind(this) }></button>
```

你可以这样优化：

```
                    <TextInput
                      onChangeValue={  this.handleValueChanging }
                    />
```

和

```
                    <button onClick={ this.handleClick }></button>
```

为了在 handleValueChanging 函数中获得正确的上下文，你可以在组件的构造函数中应用它：

```
constructor(){
            this.handleValueChanging = this.handleValueChanging.bind(this)
          }
```

更多内容请参见 binding a function passed to a component

或者你可以使用类似这样的解决方案：https://github.com/andreypopp/autobind-decorator 并简单地在你想绑定的每个方法上添加 @autobind decorator：

```
        @autobind
handleValueChanging(newValue)
        {
            //处理事件
        }
```

Chapter 30: Render Best Practises

Topic for important notes about specific Component.render method behavoir.

Section 30.1: Functions in JSX

For better performance it's important to avoid using of array (lambda) function in JSX.

As explained at https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md :

> A bind call or arrow function in a JSX prop will create a brand new function on every single render. This is bad for performance, as it will result in the garbage collector being invoked way more than is necessary. It may also cause unnecessary re-renders if a brand new function is passed as a prop to a component that uses reference equality check on the prop to determine if it should update.

So if have jsx code block like this:

```
                    <TextInput
                     onChangeValue={  value => this.handleValueChanging(value) }
                    />
```

or

```
                    <button onClick={ this.handleClick.bind(this) }></button>
```

you can make it better:

```
                    <TextInput
                      onChangeValue={  this.handleValueChanging }
                    />
```

and

```
                    <button onClick={ this.handleClick }></button>
```

For correct context within handleValueChanging function you can apply it in constructor of component:

```
          constructor(){
            this.handleValueChanging = this.handleValueChanging.bind(this)
          }
```

more in binding a function passed to a component

Or you can use solutions like this: https://github.com/andreypopp/autobind-decorator and simply add @autobind decorator to each methos that you want bind to:

```
        @autobind
        handleValueChanging(newValue)
        {
            //processing event
        }
```

第31章：调试

第31.1节：在安卓中启动远程JS调试

您可以从开发者菜单启动远程调试。选择启用远程调试后，它会打开谷歌浏览器，方便您将输出记录到控制台。您也可以在js

代码中编写调试器语法。

第31.2节：使用console.log()

您可以使用`console.log()`在终端打印日志信息。为此，请打开一个新的终端并运行以下命令（安卓）：

```
react-native log-android
```

如果您使用的是iOS，请运行以下命令：

```
react-native log-ios
```

您现在将在此终端看到所有日志信息

Chapter 31: Debugging

Section 31.1: Start Remote JS Debugging in Android

You can start the remote debugging from Developer menu. After selecting the enable remote debugging it will open Google Chrome, So that you can log the output into your console. You can also write debugger syntax into your js code.

Section 31.2: Using console.log()

You can print log message in the terminal using `console.log()`. To do so, open a new terminal and run following command for Android:

```
react-native log-android
```

or following command if you are using iOS:

```
react-native log-ios
```

You will now start to see all the log message in this terminal

第32章：单元测试

单元测试是一种低级别的测试方法，用于测试代码中最小的单元或组件。

第32.1节：使用Jest进行React Native单元测试

从react-native版本0.38开始，运行react-native init时默认包含了Jest配置。以下配置应自动添加到你的package.json文件中：

```
    "scripts": {
    "start": "node node_modules/react-native/local-cli/cli.js start",
    "test": "jest"
    },
    "jest": {
     "preset": "react-native"
    }
```

你可以运行run npm test或jest来在react native中进行测试。代码示例：Link

Chapter 32: Unit Testing

Unit testing is a low level testing practice where smallest units or components of the code are tested.

Section 32.1: Unit Test In React Native Using Jest

Starting from react-native version 0.38, a Jest setup is included by default when running react-native init. The following configuration should be automatically added to your package.json file:

```
    "scripts": {
    "start": "node node_modules/react-native/local-cli/cli.js start",
    "test": "jest"
    },
    "jest": {
     "preset": "react-native"
    }
```

You can run `run npm test or jest` to test in react native. For code example: Link

致谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供此内容，更多更改可发送至web@petercv.com以发布或更新新内容

阿卜杜勒阿齐兹·阿尔卡拉希	第12章和第18章
阿迪亚·辛格	第28章
艾哈迈德·阿尔哈达德	第27章
艾哈迈德·阿里	第5章
亚历克斯·贝莱茨	第9、15、24和30章
阿里雷扎·瓦利扎德	第15章
安德烈斯·C·维斯卡	第22章
安基特·辛哈	第25、26和32章
安东B	第15章
卡西奥·桑托斯	第20章
叫我诺姆	第3章
克里斯·佩纳	第3章和第15章
corasan	第25章
丹尼尔·施密特	第15章
大卫	第6章
德米特里·佩图霍夫	第1、14和15章
尼特皮克博士	第1章
epsilondelta	第14章
fson	第3章
加布里埃尔·迪耶斯	第16章
伊丹	第3章和第14章
贾加迪什·乌帕迪亚伊	第3、6、9、14、15、16、17和31章
吉加尔·沙阿	第4、8和17章
卡勒布·波蒂略	第1章和第11章
利龙·亚哈达夫	第5章
卢卡斯·奥利维拉	第1章
卢因·乔·缅	第18章和第21章
manosim	第1章、第14章和第20章
马耶尔	第21章
迈克尔·汉考克	第10章
迈克尔·赫尔维	第26章
迈克尔·S	第20章
莫斯塔菲兹·拉赫曼	第31章
莫扎克	第14章
Noitidart	第18章
潘卡杰·塔库尔	第26章
帕斯卡尔·勒·梅雷尔	第20章
respectTheCode	第15章
Scimonster	第1章和第8章
Serdar Değirmenci	第17章
shaN	第15章和第29章
Sriraman	第10章、第14章、第18章、第19章和第20章
stereodenis	第2章
sudo bangbang	第7、9、13和32章
特贾什维·卡尔普·塔鲁	第15章和第29章
蒂姆·里贾维奇	第6和14章
图沙尔·卡蒂瓦达	第1章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content, more changes can be sent to web@petercv.com for new content to be published or updated

Abdulaziz Alkharashi	Chapters 12 and 18
Aditya Singh	Chapter 28
Ahmed Al Haddad	Chapter 27
Ahmed Ali	Chapter 5
Alex Belets	Chapters 9, 15, 24 and 30
Alireza Valizade	Chapter 15
Andres C. Viesca	Chapter 22
Ankit Sinha	Chapters 25, 26 and 32
AntonB	Chapter 15
Cássio Santos	Chapter 20
CallMeNorm	Chapter 3
Chris Pena	Chapters 3 and 15
corasan	Chapter 25
Daniel Schmidt	Chapter 15
David	Chapter 6
Dmitry Petukhov	Chapters 1, 14 and 15
Dr. Nitpick	Chapter 1
epsilondelta	Chapter 14
fson	Chapter 3
Gabriel Diez	Chapter 16
Idan	Chapters 3 and 14
Jagadish Upadhyay	Chapters 3, 6, 9, 14, 15, 16, 17 and 31
Jigar Shah	Chapters 4, 8 and 17
Kaleb Portillo	Chapters 1 and 11
Liron Yahdav	Chapter 5
Lucas Oliveira	Chapter 1
Lwin Kyaw Myat	Chapters 18 and 21
manosim	Chapters 1, 14 and 20
Mayeul	Chapter 21
Michael Hancock	Chapter 10
Michael Helvey	Chapter 26
Michael S	Chapter 20
mostafiz rahman	Chapter 31
Mozak	Chapter 14
Noitidart	Chapter 18
Pankaj Thakur	Chapter 26
Pascal Le Merrer	Chapter 20
respectTheCode	Chapter 15
Scimonster	Chapters 1 and 8
Serdar Değirmenci	Chapter 17
shaN	Chapters 15 and 29
Sriraman	Chapters 10, 14, 18, 19 and 20
stereodenis	Chapter 2
sudo bangbang	Chapters 7, 9, 13 and 32
Tejashwi Kalp Taru	Chapters 15 and 29
Tim Rijavec	Chapters 6 and 14
Tushar Khatiwada	Chapter 1

维克托·塞č	第23章
Virat18	第20章
xhg	第1章
叶夫亨·杜比宁	第1章和第3章
扎卡里亚·里杜赫	第2章、第16章和第17章
阮振杰	第3章

Viktor Seč	Chapter 23
Virat18	Chapter 20
xhg	Chapter 1
Yevhen Dubinin	Chapters 1 and 3
Zakaria Ridouh	Chapters 2, 16 and 17
zhenjie ruan	Chapter 3