EMBAIXO DO CAPO DO REACT NATIVE

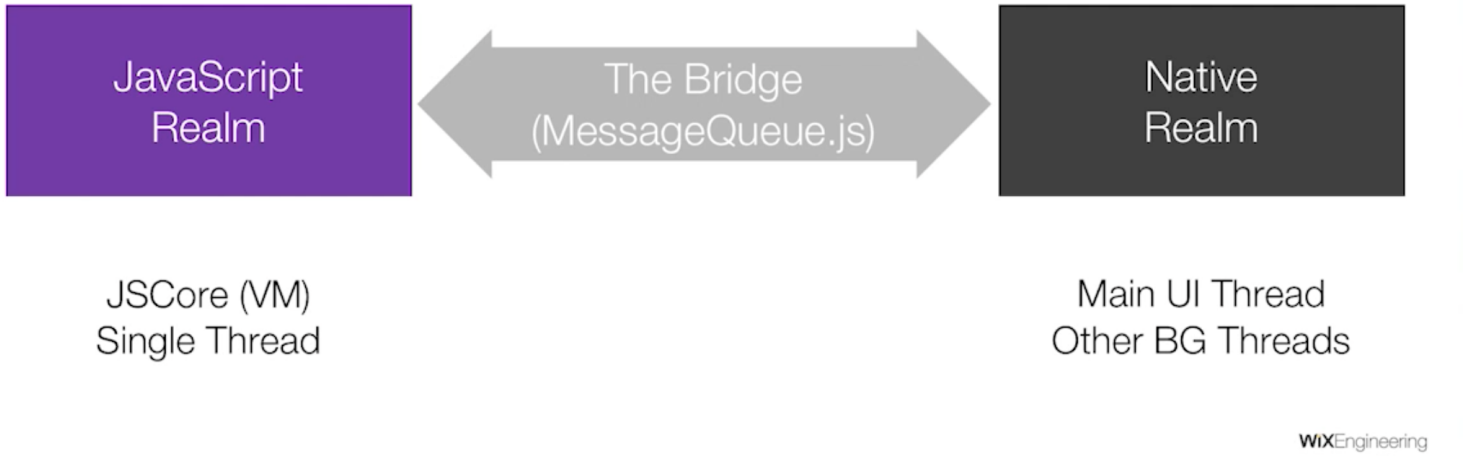
Saia da mediocridade, entenda como funciona o realmente React Native. Mais cedo ou mais tarde isso poderá lhe ajudar muito.

por Jose Urbano Duarte Junior 14/02/2018

~ 6 min. / 1119 palavras

O React Native é um framework de desenvolvimento híbrido, isto é você produz um único código Javascript que interpretado e transformado em código nativo. Componentes Estruturantes do React Native (Transformando código Javascript em Nativo)

Aqui faremos um resumo breve sobre alguns componentes básicos: o **transpilador**, o **empacotador**, o **motor**, a **brigde** e o **renderizador**.



Tempo de compilação

O **transpilador** chamado de `Babel` é o componente responsável por transpilar seu código Javascript escrito em ES6 em um código Javascript puro.

Diferentemente de um processo de compilação em que um código é transformado em linguagem de máquina, na transpilação um código legível é transformado em outro código legível.

Como é sabido, o Javascript vem evoluindo ao longo dos anos e todo ano novas definições são feitas.

Assim, o `Babel` é o cara responsável por pegar um código escrito no último padrão e transformá-lo em um código que pode ser interpretado por um motor escrito há anos atrás quando essas novas definições não existiam.

O **empacotador** chamado de `Packager` é o cara responsável por combinar seus diversos arquivos Javascript em um arquivo único.

Depois que babel deixou o código em um formato compatível com qualquer motor Javascript, o `Packager` mimifica os seus arquivos em um só.

Nesse momento é como se você tivesse escrito todo seu projeto em um único arquivo.

Isso pode parecer estranho, mas facilita para o trabalho do motor. Afinal, é muito mais fácil buscar e ler um arquivo do que ficar manipulando diversos arquivos.

Tempo de execução

O **motor JavaScriptCore** é o cara responsável por entender e processar o código javascript.

Esse `motor JavaScriptCore` já existe no código nativo sendo utilizado pelo Safari para processamento do javascript.

Assim, sua aplicação React Native só precisa pegar seu bundle (código empacotado) e entregar para o `motor JavaScriptCore`.

Se você estudar React Native a fundo, você verá que esse bundle é colocado dentro do Aplicativo nativo como se fosse uma imagem ou um asset qualquer.

Depois o caminho para o arquivo é repassado para que o `motor JavaScriptCore` comece a processar o código.

A **Bridge** é o barramento utilizado para comunicação entre o `motor JavaScriptCore` e o `Renderizador`. Exemplo: o seu código está sendo executado no `motor JavaScriptCore` para que ele solicite que o nativo crie uma tela.

Essa informação é passada do motor para o renderizado pela `Bridge`. Entretanto, a informação também pode percorrer o caminho inverso.

Quando um botão é clicado, o clique chega primeiro para o nativo. O renderizado nativo, então, envia esse clique para o `motor JavaScriptCore`.

O **Renderizador** é o componente responsável para transformar as mensagens vindas da Brigde em componentes visuais. Assim, a `Bridge` envia um pedido:

```
Message 1: “Crie um botão de cor azul e com texto ABC”
```

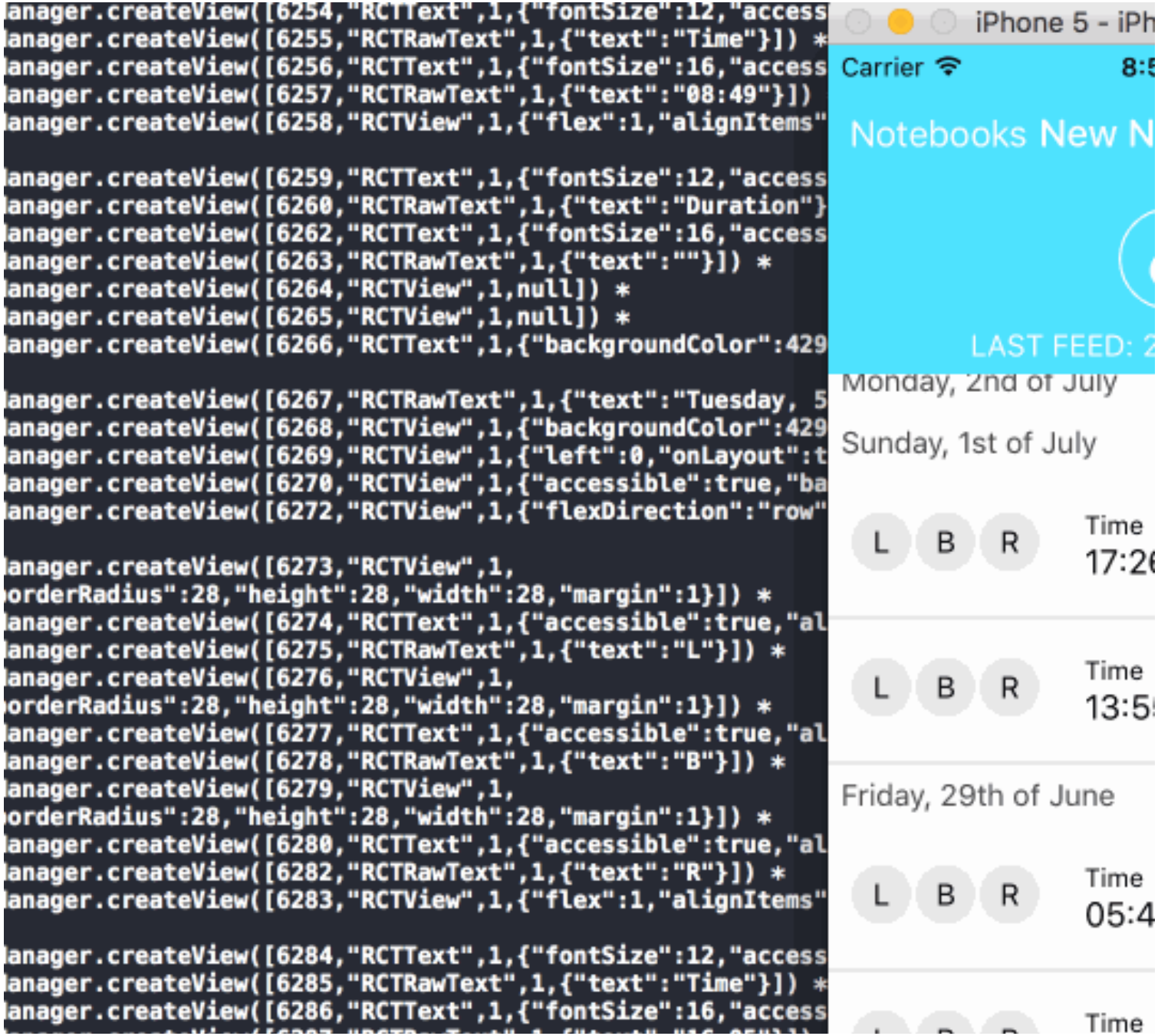
O `Renderizador` chama o método nativo:

```
Action 1: CreateButton. (Android)
Action 1: CreateViewButton. (IOS)
```

Como o `Renderizador` é nativo e implementado nas duas plataformas, uma mensagem como a mensagem

Na verdade essa é mágica do React Native. Os componentes são nativos e desenhados, conforme o padrão de cada plataforma.

A imagem abaixo demonstra a comunicação passando da `Bridge` para o `Renderizador`. Observe como ao rolar a tela, novas mensagens solicitando a criação de novos componentes são enviadas do Javascript para o Nativo.


anager.createView([6254,"RCTText",1,{"fontSize":12,"access
anager.createView([6255,"RCTRawText",1,{"text":"Time"}]) *
anager.createView([6256,"RCTText",1,{"fontSize":16,"access
anager.createView([6257,"RCTRawText",1,{"text":"08:49"}])
anager.createView([6258,"RCTView",1,{"flex":1,"alignItems"
anager.createView([6259,"RCTText",1,{"fontSize":12,"access
anager.createView([6260,"RCTRawText",1,{"text":"Duration"}
anager.createView([6262,"RCTText",1,{"fontSize":16,"access
anager.createView([6263,"RCTRawText",1,{"text":""}]) *
anager.createView([6264,"RCTView",1,null]) *
anager.createView([6265,"RCTView",1,null]) *
anager.createView([6266,"RCTText",1,{"backgroundColor":429
anager.createView([6267,"RCTRawText",1,{"text":"Tuesday, 5
anager.createView([6268,"RCTView",1,{"backgroundColor":429
anager.createView([6269,"RCTView",1,{"left":0,"onLayout":t
anager.createView([6270,"RCTView",1,{"accessible":true,"ba
anager.createView([6272,"RCTView",1,{"flexDirection":"row"
anager.createView([6273,"RCTView",1,
orderRadius":28,"height":28,"width":28,"margin":1}]) *
anager.createView([6274,"RCTText",1,{"accessible":true,"al
anager.createView([6275,"RCTRawText",1,{"text":"L"}]) *
anager.createView([6276,"RCTView",1,
orderRadius":28,"height":28,"width":28,"margin":1}]) *
anager.createView([6277,"RCTText",1,{"accessible":true,"al
anager.createView([6278,"RCTRawText",1,{"text":"B"}]) *
anager.createView([6279,"RCTView",1,
orderRadius":28,"height":28,"width":28,"margin":1}]) *
anager.createView([6280,"RCTText",1,{"accessible":true,"al
anager.createView([6282,"RCTRawText",1,{"text":"R"}]) *
anager.createView([6283,"RCTView",1,{"flex":1,"alignItems"
anager.createView([6284,"RCTText",1,{"fontSize":12,"access
anager.createView([6285,"RCTRawText",1,{"text":"Time"}]) *
anager.createView([6286,"RCTText",1,{"fontSize":16,"access
iPhone 5 - iPh
Carrier
8:5
Notebooks New N
LAST FEED: 2
Monday, 2nd of July
Sunday, 1st of July
L B R
Time
17:26
L B R
Time
13:5
Friday, 29th of June
L B R
Time
05:4
Time

Gargalos

Um questionamento natural ao entender a arquitetura do React Native é quanto a performance. O React Native possui um gargalo: a `Brigde`. É na `Bridge` que passa toda informação entre o `motor` e o `renderizador`. Assim, passar objetos muito grandes pela `Bridge` causa uma sobrecarga.

Virtual DOM

Já foi explicado que a `Bridge` é um dos gargalos do React Native, por isso seus arquitetos tentaram mitigar esse problema utilizando a Virtual DOM. O Virtual DOM é um árvore virtual que representa o estado de cada componente na tela. Assim, ao invés de passar toda e qualquer modificação que aconteça em um componente para o `renderizador`, o Virtual DOM compara novo componente com o estado antigo do DOM Virtual e envia para `Bridge` somente as informações que foram modificadas.

Exemplo: Imagine que você tenha 20 componentes na tela e uma funcionalidade que altere o valor de um texto ao se clicar em 1 botão. Ao invés de enviar para o renderizador toda a tela com o novo texto, o Virtual DOM filtra e envia apenas o modificado componente.

Atualizando o seu App Remotamente

Não é necessário entender os componentes estruturantes do React para fazer um app, mas isso lhe ajudará a

Uma das vantagens claras do React Native é possibilidade de se atualizar um aplicativo sem necessidade de passar pelos longos e burocráticos processos das lojas (Apple e Google).

Se você entendeu bem a parte do empacotamento, você terá vislumbrado a possibilidade de receber um javascript já mimificado pela internet e solicitar que o motor faça a interpretação desse código.

Basicamente você pode utilizar os componentes 1,2 (`Babel` e `Packager`) para gerar um novo bundle e substituir seu bundle anterior pelo novo. Projetos como o CodePush da Microsoft fazem exatamente isso.

Limitações

Claro que existem limitações. Todo aplicativo é como uma sandbox com permissões definidas e autorizadas pelo usuário.

Assim, se seu aplicativo nativo não declarou que faria uso da câmera no momento que foi enviado para Google, não adiantará você mandar uma atualização que tente fazer este uso.

O lado nativo, que não pode ser atualizado remotamente, irá negar acesso ao recurso. Além disso, se você fizer mal uso desse recurso, a Apple e a Google podem acabar banindo seu aplicativo e/ou a sua conta de desenvolvedor.

Outras limitações associadas ao fato do lado nativo ser fixo, é que seu novo bundle javascript não introduz novas dependências que façam usam de bibliotecas nativas.

Exemplo 1: Não é possível gerar um bundle utilizando o `react-native@0.51.0` e, posteriormente, atualizar remotamente seu app com um novo bundle que utiliza o `react-native@0.52.0`. Isso porque a implementação React possui código nativo (renderizador), assim o seu bundle `react-native@0.52.0` estaria tentando conversar com um renderizador `react-native@0.51.0`.

Exemplo 2: Não é possível gerar um bundle utilizando a biblioteca `react-native-camera@1.0.0` e depois atualizar seu app remotamente com um bundle que utilize a `react-native-camera@1.1.0`. Isso porque a biblioteca `react-native-camera` utiliza o comando react-native link para sua instalação. Esse comando injeta código nativo no aplicativo. Assim, como tivemos no exemplo 1, teríamos uma biblioteca 1.1 tentando conversar com código nativo feito para biblioteca 1.0

Exemplo 3: É possível gerar um bundle utilizando a biblioteca `react-native-mobx@1.0.0` e depois atualizar seu app remotamente com um bundle que utilize a `react-native-mobx@1.1.0`. Isso porque a biblioteca `react-native-mobx` é puramente javascript. Ela não executa o comando react-native link. Todo seu código será interpretado pelo motor javascript.

Referências para sobre o assunto

https://www.youtube.com/watch?v=NkbO-Vhqbl0&t=589s

https://www.youtube.com/watch?v=hDviGU-57lU&t=1315s

https://www.youtube.com/watch?v=Ah2qNbI40vE&t=471s

- Indo além no Mundo React-Native
- Componentes além do reuso
- Parcel Bundler: Criando um projeto React
- Fontes customizadas no React-Native
- Ember.js: Deploy no Heroku

ALSO ON **TABLELESS**

Elastic Stack + Beats e como tudo isso ...

4 anos atrás · 3 comentários

O que é o Elastic Stack e o que são os Beats

Comentário: Se compare com os outros

4 anos atrás · 2 comentários

Essa foi uma série de tweets que fiz outro dia. No final, tem o link da thread. ...

Guia fácil para usar localStorage com ...

4 anos atrás · 6 comentários

Entendendo um pouco sobre como utilizar localStorage com Javascript

Tab Sel

4 an

Tab sele

Você vai gostar de ler:

Indo além no Mundo React-Native

Parcel Bundler: Criando um projeto React

Fontes customizadas no React-Native

Ember.js: Deploy no Heroku

Categorias

tecnologia-e-tendências (316)

geral (297)

artigos (290)

técnicas-e-práticas (269)

javascript (264)

código (214)

browsers (206)

html (191)

css (183)

mercado (122)

SOBRE

SOBRE O TABLELESS

CONTATO

ANUNCIE

SEJA UM AUTOR

FAZEMOS CÓDIGO FRONT-END

ACOMPANHE

WEBINARS

CANAL NO MEDIUM

CANAL NO TELEGRAM

FEMUG

MEETUPCSS

PODCAST ZOFE

BRAZILJS

DEVNAESTRADA

FRONT-END BRASIL

Escrito pela e para a comunidade web brasileira. Ajude.