# 0403-讲义-RN原理与介绍



## 搞过 RN 的 扣 1

## 课程目标（跨端 / RN）

P6：

- 会使用 RN，了解RN同类别的产品，了解移动端的主要技术方案，有一定的跨端开发经验，踩过一些坑；

P6+ ～ P7：

- 知道如何与native进行数据交互，知道ios与安卓jsbridge实现原理。
- 知道移动端webview和基础能力，包括但不限于：webview资源加载优化方案；webview池管理、

独立进程方案；native路由等。

- 能够给出完整的前后端对用户体系的整体技术架构设计，满足多业务形态用户体系统一。考虑跨域名、多组织架构、跨端、用户态开放等场景。(BFF， 技术栈选型。。)

- 把react以及跨端相关的知识点，一起串一下。
  - 在窥探 react- native 原理的同时，给大家总结一下 react 框架上的一些知识，同时，也带大家一起了解一些 跨端编译方面的知识。

# 课程大纲（本节）

- **个人知识图谱 15'；**
- **JS rising Star 20'；**
- **Hybrid 架构演进方案 65';**
- **React Native 和 React 之间的关系；100'；**

- 介绍 RN 的背景，与其他跨端开发之间的异同；
- 介绍 RN 的渲染模式，对比到 React 框架、小程序框架；
- 介绍 RN 的整体原理。

## 为什么要做跨端

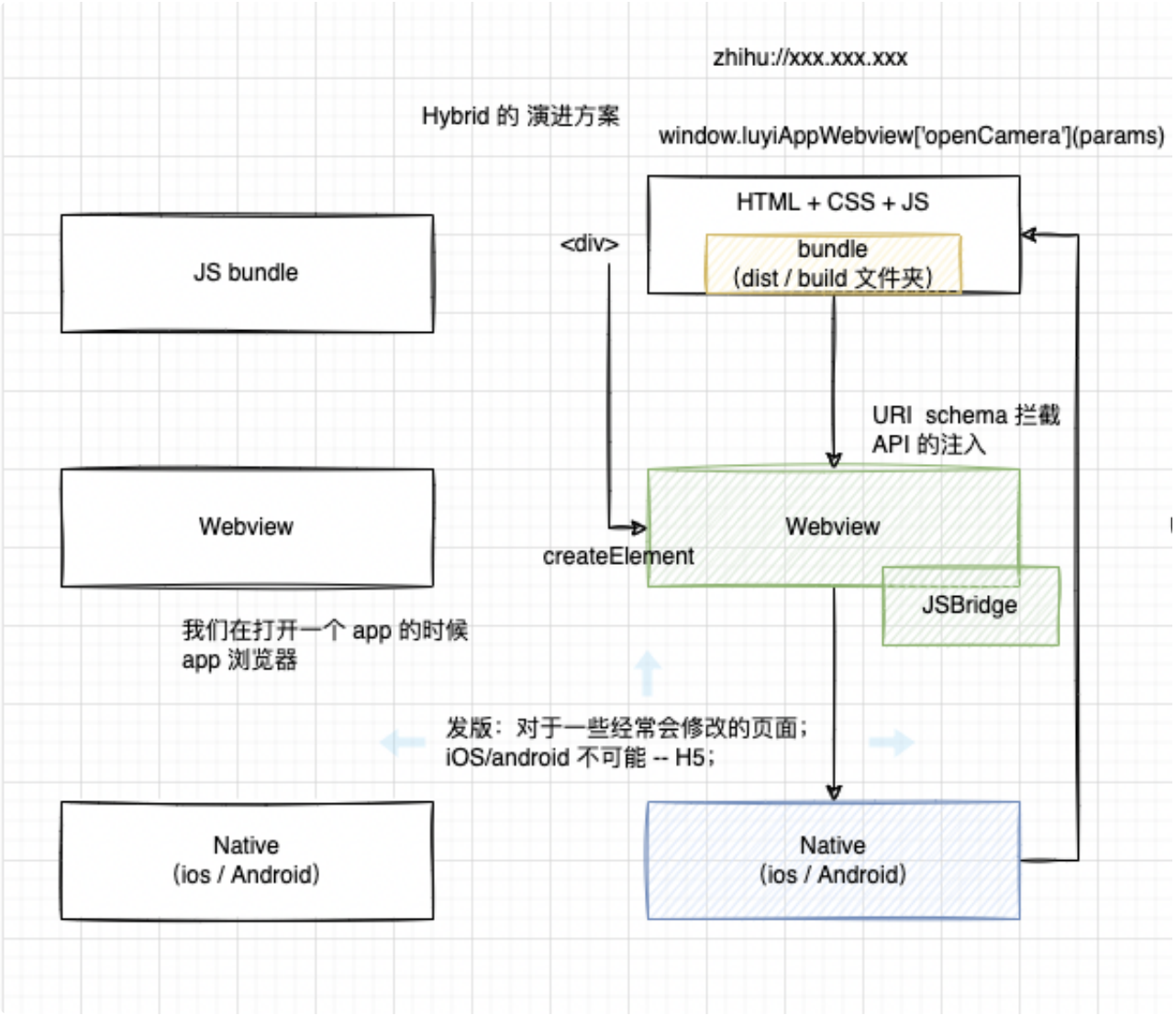
### 个人知识图谱

RN

在 2021 JavaScript Rising Star： 链接

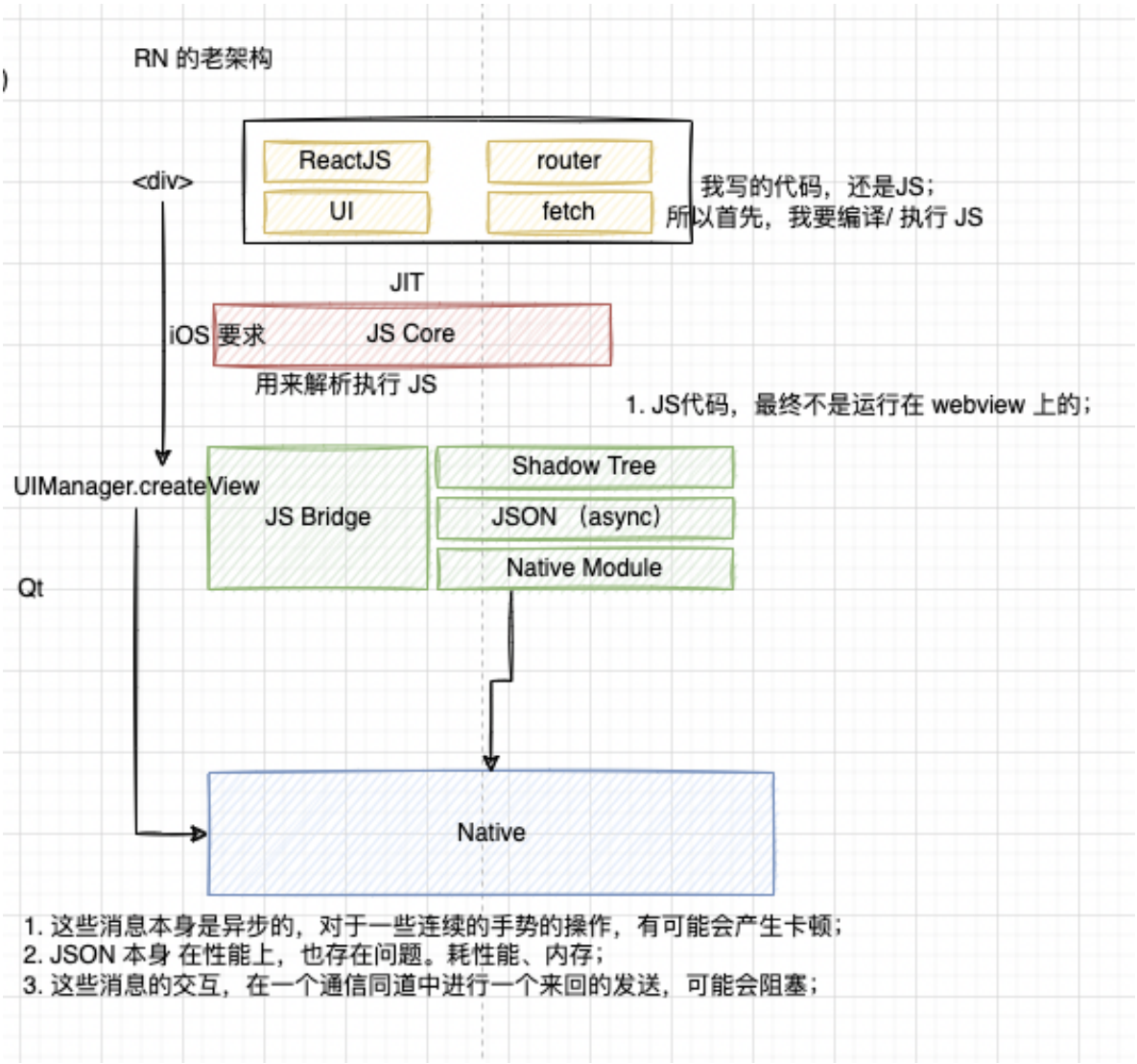
## 移动端平台方案演进

### 1. hybrid


zhihu://xxx.xxx.xxx
Hybrid 的 演进方案
window.luyiAppWebview['openCamera'](params)
HTML + CSS + JS
bundle
(dist / build 文件夹)
JS bundle

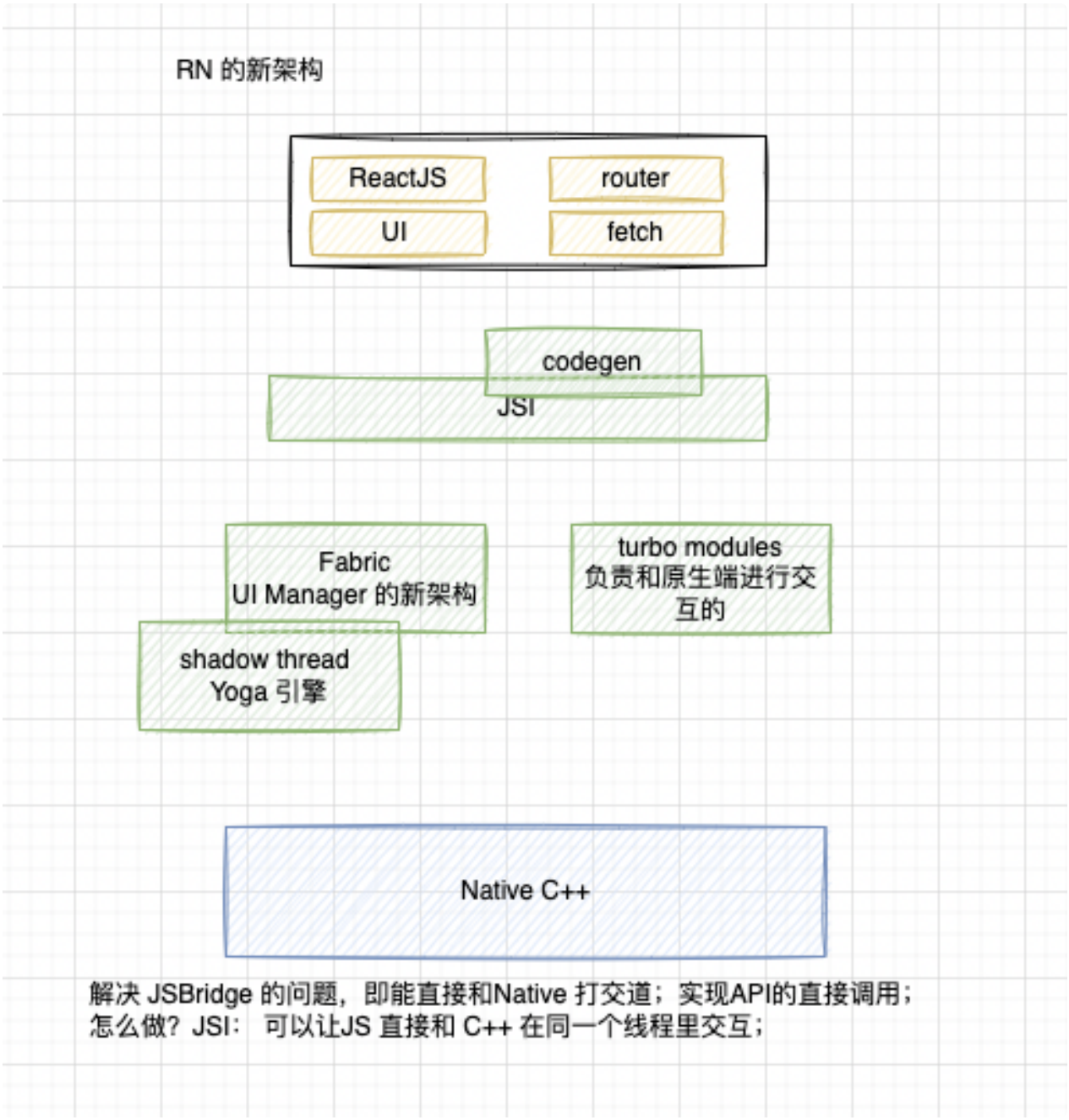
URI schema 拦截
API 的注入
Webview
Webview
createElement
JSBridge
我们在打开一个 app 的时候
app 浏览器
发版：对于一些经常会修改的页面；
iOS/android 不可能 -- H5；
Native
(ios / Android)
Native
(ios / Android)

## 2. RN 老架构


RN 的老架构
ReactJS
router
UI
fetch
我写的代码，还是JS；
所以首先，我要编译/ 执行 JS

JIT
iOS 要求
JS Core
用来解析执行 JS
1. JS代码，最终不是运行在 webview 上的；
UIManager.createView
JS Bridge
Shadow Tree
JSON （async）
Native Module
Qt
Native
1. 这些消息本身是异步的，对于一些连续的手势的操作，有可能会产生卡顿；
2. JSON 本身 在性能上，也存在问题。耗性能、内存；
3. 这些消息的交互，在一个通信同道中进行一个来回的发送，可能会阻塞；

### 3. RN 新架构




**4. 其他...**

## RN 的原理

### React的设计理念

在运行时开发者能够处理 React JSX 的核心基础其实在于 **React 的设计理念**，React 将自身能力充分解耦，并提供给社区接入关键环节。这里我们需要先进行一些 React 原理解析。

React 的整体设计理念可以分为三个部分：

- React Core
- React Renderer
- Reconciler
- 在这里我们需要了解的是：

自定义 renderer --- 宿主配置 **hostConfig** --- React reconciler --- react core

JavaScript | 复制代码

```
HostConfig.getPublicInstance
HostConfig.getRootHostContext
HostConfig.getChildHostContext

HostConfig.prepareForCommit

HostConfig.resetAfterCommit

HostConfig.createInstance

HostConfig.appendInitialChild

HostConfig.finalizeInitialChildren

HostConfig.prepareUpdate
HostConfig.shouldSetTextContent
HostConfig.shouldDeprioritizeSubtree

HostConfig.createTextInstance

HostConfig.scheduleDeferredCallback
HostConfig.cancelDeferredCallback
HostConfig.setTimeout
HostConfig.clearTimeout
HostConfig.noTimeout
HostConfig.now
HostConfig.isPrimaryRenderer
HostConfig.supportsMutation
HostConfig.supportsPersistence
HostConfig.supportsHydration
// -------------------
//      Mutation
//     (optional)
// -------------------
HostConfig.appendChild
HostConfig.appendChildToContainer
HostConfig.commitTextUpdate
HostConfig.commitMount
HostConfig.commitUpdate
HostConfig.insertBefore
HostConfig.insertInContainerBefore
HostConfig.removeChild
HostConfig.removeChildFromContainer
HostConfig.resetTextContent
HostConfig.hideInstance
```

```
46   HostConfig.hideTextInstance
47   HostConfig.unhideInstance
48   HostConfig.unhideTextInstance
49   // -------------------
50   //     Persistence
51   //     (optional)
52   // -------------------
53   HostConfig.cloneInstance
54   HostConfig.createContainerChildSet
55   HostConfig.appendChildToContainerChildSet
56   HostConfig.finalizeContainerChildren
57   HostConfig.replaceContainerChildren
58   HostConfig.cloneHiddenInstance
59   HostConfig.cloneUnhiddenInstance
60   HostConfig.createHiddenTextInstance
61   // -------------------
62   //     Hydration
63   //     (optional)
64   // -------------------
65   HostConfig.canHydrateInstance
66   HostConfig.canHydrateTextInstance
67   HostConfig.getNextHydratableSibling
68   HostConfig.getFirstHydratableChild
69   HostConfig.hydrateInstance
70   HostConfig.hydrateTextInstance
71   HostConfig.didNotMatchHydratedContainerTextInstance
72   HostConfig.didNotMatchHydratedTextInstance
73   HostConfig.didNotHydrateContainerInstance
74   HostConfig.didNotHydrateInstance
75   HostConfig.didNotFindHydratableContainerInstance
76   HostConfig.didNotFindHydratableContainerTextInstance
77   HostConfig.didNotFindHydratableInstance
78   HostConfig.didNotFindHydratableTextInstance
```

### RN 的原理

#### 注册与发布

`AppRegistry` 是所有 React Native 应用的 JS 入口。应用的根组件应当通过 `AppRegistry.registerComponent` 方法注册自己，然后原生系统才可以加载应用的代码包并且在启动完成之后通过调用 `AppRegistry.runApplication` 来真正运行应用。

```javascript
AppRegistry.registerComponent(appName, () => App);
```

Libraries/ReactNative/AppRegistry.js

JavaScript 复制代码

```
  registerComponent(
      appKey: string,
      componentProvider: ComponentProvider,
      section?: boolean,
    ): string {
      let scopedPerformanceLogger = createPerformanceLogger();
      runnables[appKey] = {
        componentProvider,
        run: (appParameters, displayMode) => {
          const concurrentRootEnabled =
            appParameters.initialProps?.concurrentRoot ||
            appParameters.concurrentRoot;
          /**********************/
          renderApplication(
            componentProviderInstrumentationHook(
              componentProvider,
              scopedPerformanceLogger,
            ),
            appParameters.initialProps,
            appParameters.rootTag,
            wrapperComponentProvider &&
  wrapperComponentProvider(appParameters),
            appParameters.fabric,
            showArchitectureIndicator,
            scopedPerformanceLogger,
            appKey === 'LogBox',
            appKey,
            coerceDisplayMode(displayMode),
            concurrentRootEnabled,
          );
        },
      };
      if (section) {
        sections[appKey] = runnables[appKey];
      }
      return appKey;
    },


    runApplication(
      appKey: string,
      appParameters: any,
      displayMode?: number,
    ): void {
      if (appKey !== 'LogBox') {
```

```
45          const logParams = __DEV__
46            ? '" with ' + JSON.stringify(appParameters)
47            : '';
48          const msg = 'Running "' + appKey + logParams;
49          infoLog(msg);
50          BugReporting.addSource(
51            'AppRegistry.runApplication' + runCount++,
52            () => msg,
53          );
54        }
55        invariant(
56          runnables[appKey] && runnables[appKey].run,
57          `"${appKey}" has not been registered. This can happen if:\n` +
58            '* Metro (the local dev server) is run from the wrong folder. ' +
59            'Check if Metro is running, stop it and restart it in the current
    project.\n' +
60            "* A module failed to load due to an error and
    `AppRegistry.registerComponent` wasn't called.",
61        );
62
63        SceneTracker.setActiveScene({name: appKey});
64        runnables[appKey].run(appParameters, displayMode);
65      },
```

Libraries/ReactNative/renderApplication.js

JavaScript 复制代码

```
function renderApplication<Props: Object>(
  RootComponent: React.ComponentType<Props>,
  initialProps: Props,
  rootTag: any,
  WrapperComponent?: ?React.ComponentType<any>,
  fabric?: boolean,
  showArchitectureIndicator?: boolean,
  scopedPerformanceLogger?: IPerformanceLogger,
  isLogBox?: boolean,
  debugName?: string,
  displayMode?: ?DisplayModeType,
  useConcurrentRoot?: boolean,
) {
  invariant(rootTag, 'Expect to have a valid rootTag, instead got ', rootTag);

  const performanceLogger = scopedPerformanceLogger ?? GlobalPerformanceLogger;

  let renderable = (
    <PerformanceLoggerContext.Provider value={performanceLogger}>
      <AppContainer
        rootTag={rootTag}
        fabric={fabric}
        showArchitectureIndicator={showArchitectureIndicator}
        WrapperComponent={WrapperComponent}
        initialProps={initialProps ?? Object.freeze({})}
        internal_excludeLogBox={isLogBox}>
        <RootComponent {...initialProps} rootTag={rootTag} />
      </AppContainer>
    </PerformanceLoggerContext.Provider>
  );

  if (__DEV__ && debugName) {
    const RootComponentWithMeaningfulName = getCachedComponentWithDebugName(
      `${debugName}(RootComponent)`,
    );
    renderable = (
      <RootComponentWithMeaningfulName>
        {renderable}
      </RootComponentWithMeaningfulName>
    );
  }

```

```
    performanceLogger.startTimespan('renderApplication_React_render');
    performanceLogger.setExtra('usedReactFabric', fabric ? '1' : '0');
    if (fabric) {
      require('../Renderer/shims/ReactFabric').render(
        renderable,
        rootTag,
        null,
        useConcurrentRoot,
      );
    } else {
      /*****************************/
      require('../Renderer/shims/ReactNative').render(renderable, rootTag);
    }
    performanceLogger.stopTimespan('renderApplication_React_render');
  }
```

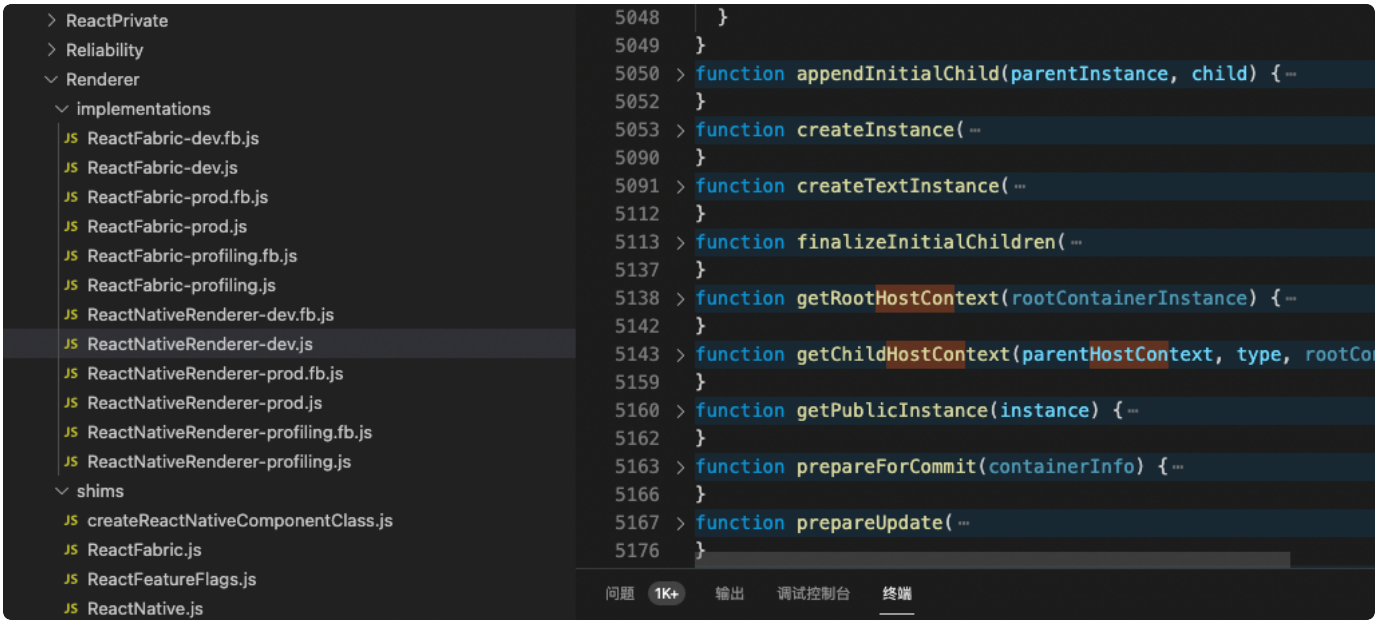
Libraries/Renderer/implementations/ReactNativeRenderer-dev.js

JavaScript | 复制代码

```
// 22976
function render(element, containerTag, callback) {
  var root = roots.get(containerTag);

  if (!root) {
    // TODO (bvaughn): If we decide to keep the wrapper component,
    // We could create a wrapper for containerTag as well to reduce special casing.
    root = createContainer(containerTag, LegacyRoot, false, null, false);
    roots.set(containerTag, root);
  }

  updateContainer(element, root, null, callback); // $FlowIssue Flow has hardcoded values for React DOM that don't work with RN

  return getPublicRootInstance(root);
}
// updateContainer
// scheduleUpdateOnFiber
// performSyncWorkOnRoot
// renderRootSync
// workLoopSync
// performUnitOfWork
// completeWork
// -HostComponent-createInstance
// -> 一直走到 createInstance

function createInstance(
  type,
  props,
  rootContainerInstance,
  hostContext,
  internalInstanceHandle
) {
  var tag = allocateTag();
  var viewConfig = getViewConfigForType(type);

  {
    for (var key in viewConfig.validAttributes) {
      if (props.hasOwnProperty(key)) {
        ReactNativePrivateInterface.deepFreezeAndThrowOnMutationInDev(
          props[key]
        );
      }
    }
```

```
    }

    var updatePayload = create(props, viewConfig.validAttributes);
      /*************************************/
    ReactNativePrivateInterface.UIManager.createView(
      tag, // reactTag
      viewConfig.uiViewClassName, // viewName
      rootContainerInstance, // rootTag
      updatePayload // props
    );
    var component = new ReactNativeFiberHostComponent(
      tag,
      viewConfig,
      internalInstanceHandle
    );
    precacheFiberNode(internalInstanceHandle, tag);
    updateFiberProps(tag, props); // Not sure how to avoid this cast. Flow
  is okay if the component is defined
    // in the same file but if it's external it can't see the types.

    return component;
  }
```

Renderer




## 一起实现一个 Render

其他的可参考资料：

附：

Taro 的包：https://github.com/NervJS/taro/tree/next/packages/taro-react

native的包：https://github.com/facebook/react/blob/main/packages/react-native-renderer/src/ReactNativeHostConfig.js

Dom 的包：https://github.com/facebook/react/blob/main/packages/react-dom/src/client/ReactDOMHostConfig.js

ART 的包：https://github.com/facebook/react/blob/main/packages/react-art/src/ReactARTHostConfig.js

jsi & jsc
https://github.com/react-native-community/discussions-and-proposals/issues/91
rn架构：
https://formidable.com/blog/2019/react-codegen-part-1/
https://formidable.com/blog/2019/jsi-jsc-part-2/

https://www.awesome-react-native.com/