React-Native 入门与原理介绍

课程总目标（两节）

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：rem 1px 方案，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。

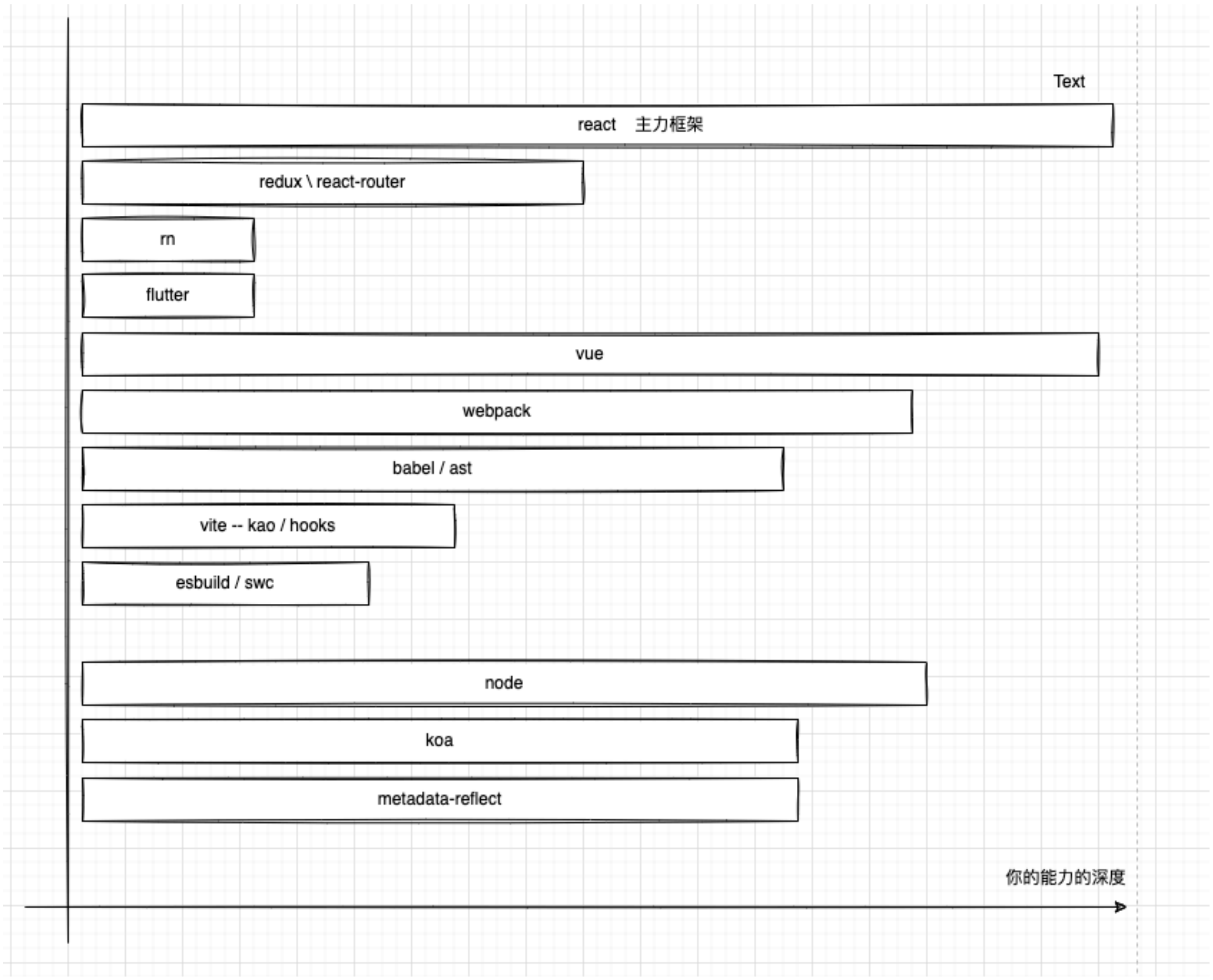
其他目标：

- 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）

- 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。

主要内容



移动端跨平台开发方案的演进

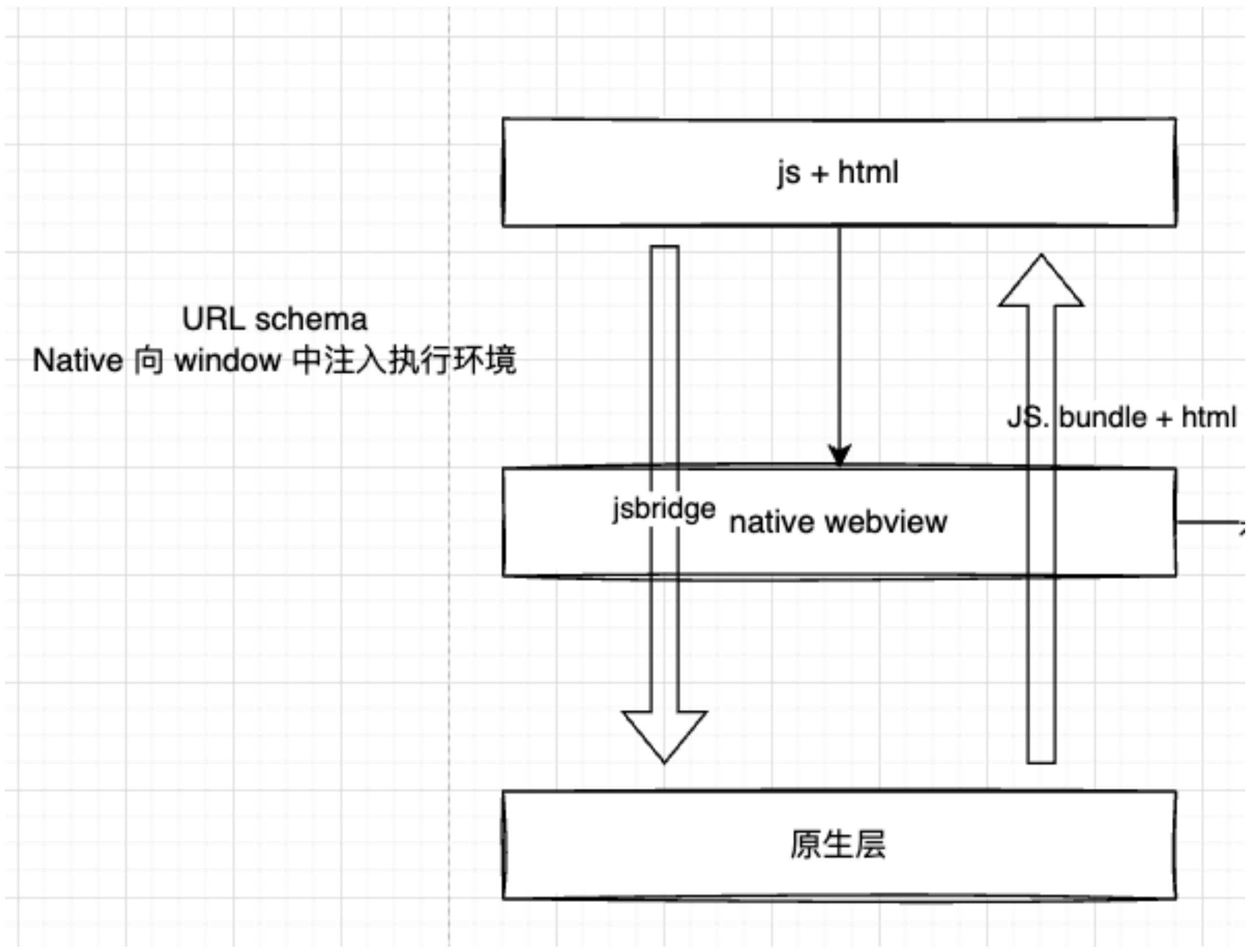
当前热点

在 2021 JavaScript Rising Star： 链接

- **React Native**
- Ionic
- **Expo**
- Quasar
- Flipper
- Flutter

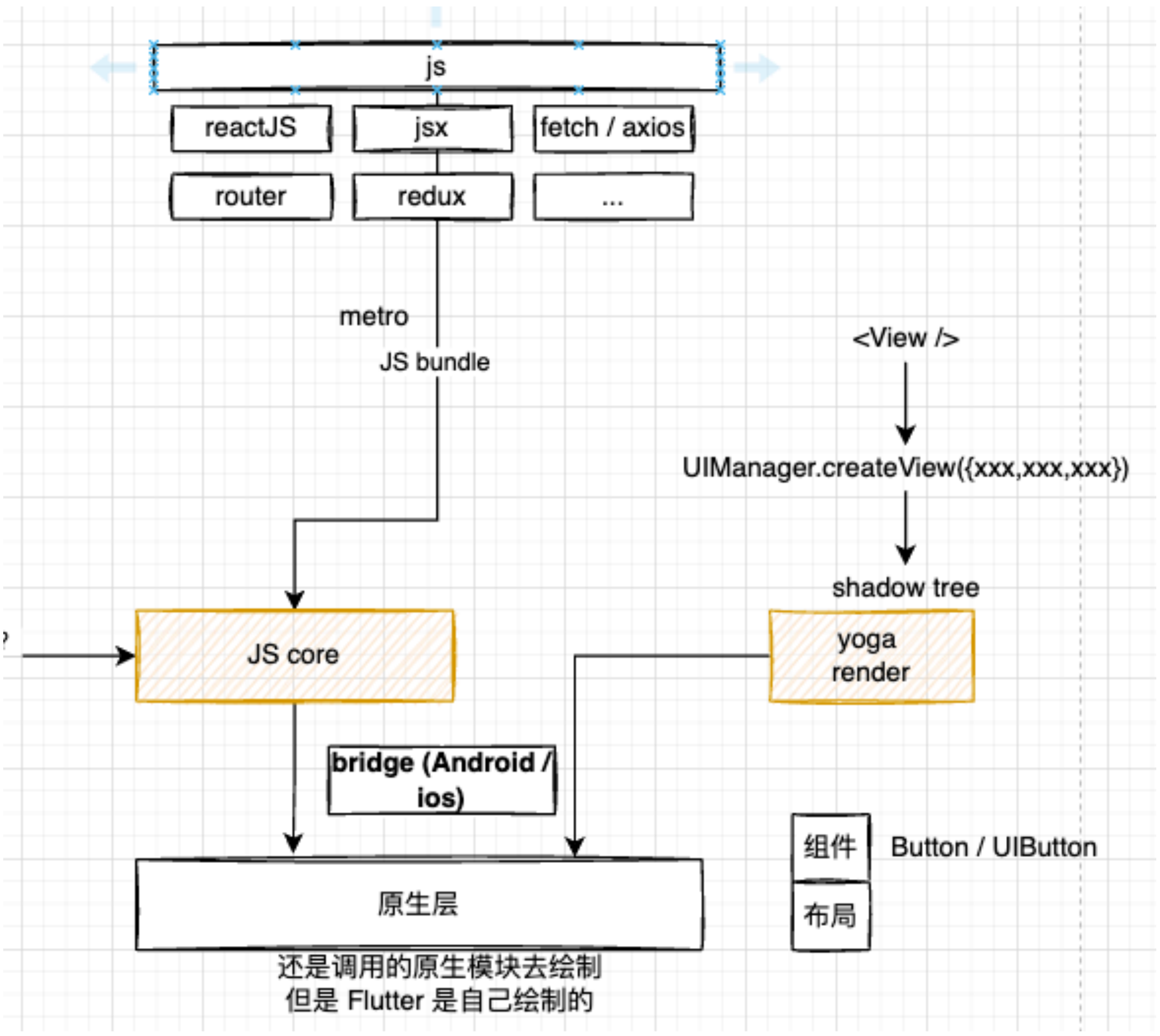
演进历史

- Webview
    - URL schema -> 如果想用 web 调用一个 windows 的程序；
        - `luyi://xxx.xxx?params1=xxx&params=2`；
        - Jssdk -- 微信小程序的 `jsbridge`；

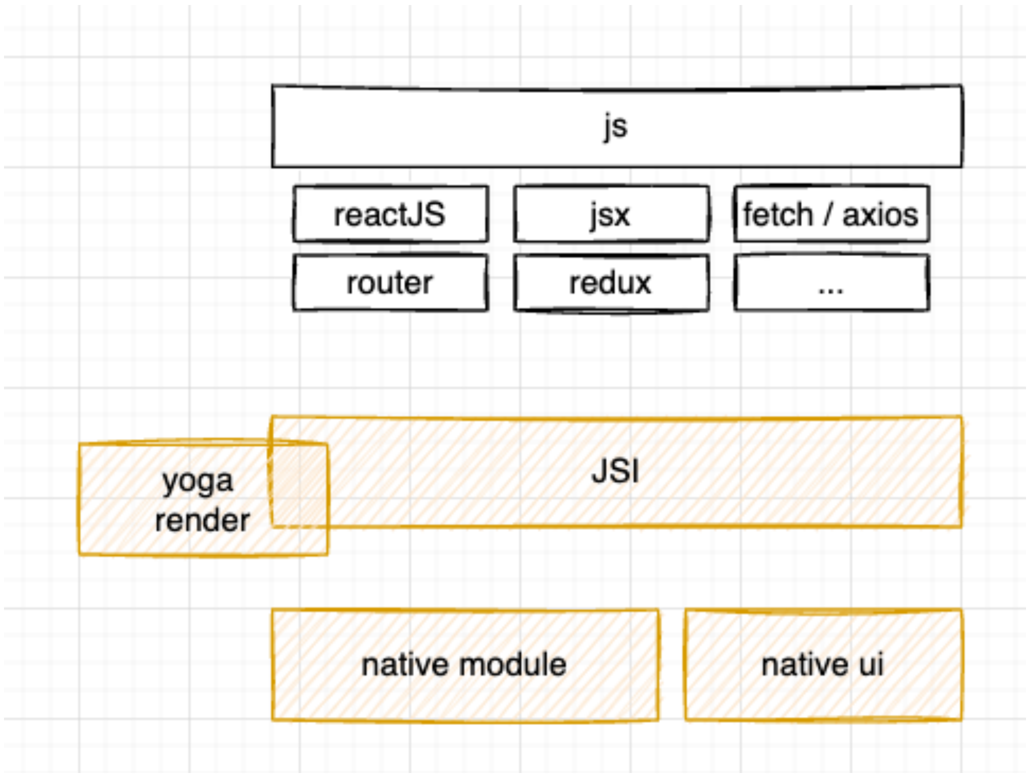


ReactNative （Weex）

- 他的目标不是一次编写，到处执行；而是，一次学习多处开发。不同平台上编写都是基于 React 的代码；
- Android 和 iOS 是有区别的，而且区别也比较大；
  - Web view： 嵌入式浏览器；
  - JavaScript Core：
    - V8 -- chrome
    - spyderMonkey -- firefox
    - JS core -- safari



- Flutter
  - 1. flutter，他是直接基于原生的Native，有一个自己的渲染引擎，skia；
    2. flutter 在这种 iOS / android 端的一致性，其实要比 RN 好；
    3. flutter dart2，学习需要一定的成本；
- RN + Fabric



端与跨端

- 端：数据获取、状态管理、页面渲染。(FE三板斧)
- 跨端：虚拟机、渲染引擎、原生交互、开发环境。

1. 数据获取

还是老三样：fetch \ axios \ XHR

2. 状态管理

React 中的 state 模式：redux / mobx

3. 页面渲染

vDOM -> Fiber -> DOM

vDom -> yoga -> iOS / android / DOM APIs -> iOS / android / Web

4. 虚拟机

RN:

- JSC - Objective-c / JS
- 到原生层，用了大量的 bridge

Flutter:

- JIT(dev) + AOT(prod)
- Skia
- 生态问题

5. 渲染引擎

yoga

6. 原生交互

Jsbridge

7. 开发环境

Web storm / VS code

- RN: Android SDK / Xcode
- React: node 浏览器

RN 的原理

React 的设计理念

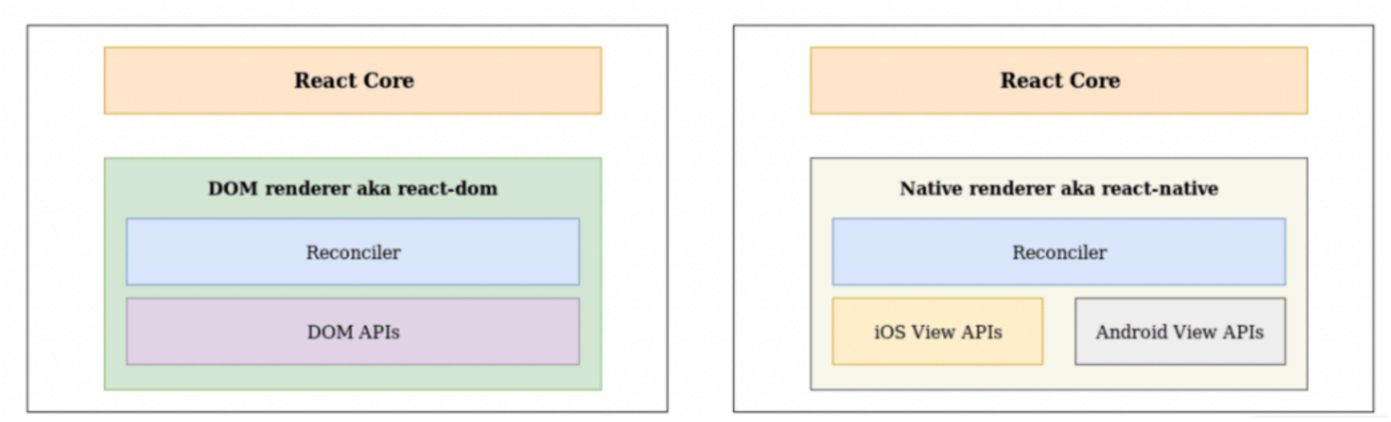
在运行时开发者能够处理 React JSX 的核心基础其实在于 **React 的设计理念**，React 将自身能力充分解耦，并提供给社区接入关键环节。这里我们需要先进行一些 React 原理解析。

React 的整体设计理念可以分为三个部分：

- React Core： 处理最核心的 APIs，与终端平台和渲染结构，主要提供了以下几个方面的能力：
  - React.createElement();
  - React.createClass();
  - React.Component;
  - React.Children
  - React.Proptypes
- React Renderer：渲染器定义了一个 React Tree 如何构建接轨不同平台，比如：
  - React-dom 渲染组件树 为 DOM elements
  - **React Native 渲染组件树为 不同原生平台视图**
- Reconciler：负责diff 算法，patch行为，可以被 React-dom、React-Native、React-ART这些 renderer 公用，并提供基础计算能力：
  - Stack reconciler - 15以及早期版本；
  - Fiber reconciler 新一代的架构；
- 在这里我们需要了解的是：

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

```
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
HostConfig.hideTextInstance
HostConfig.unhideInstance
HostConfig.unhideTextInstance
// -------------------
//     Persistence
//     (optional)
// -------------------
HostConfig.cloneInstance
HostConfig.createContainerChildSet
HostConfig.appendChildToContainerChildSet
HostConfig.finalizeContainerChildren
HostConfig.replaceContainerChildren
HostConfig.cloneHiddenInstance
HostConfig.cloneUnhiddenInstance
HostConfig.createHiddenTextInstance
// -------------------
//     Hydration
//     (optional)
// -------------------
HostConfig.canHydrateInstance
HostConfig.canHydrateTextInstance
HostConfig.getNextHydratableSibling
HostConfig.getFirstHydratableChild
HostConfig.hydrateInstance
HostConfig.hydrateTextInstance
HostConfig.didNotMatchHydratedContainerTextInstance
HostConfig.didNotMatchHydratedTextInstance
HostConfig.didNotHydrateContainerInstance
HostConfig.didNotHydrateInstance
HostConfig.didNotFindHydratableContainerInstance
HostConfig.didNotFindHydratableContainerTextInstance
HostConfig.didNotFindHydratableInstance
HostConfig.didNotFindHydratableTextInstance
```



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

RN 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。

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

Libraries/ReactNative/AppRegistry.js

```
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') {
      const logParams = __DEV__
        ? '" with ' + JSON.stringify(appParameters)
        : '';
      const msg = 'Running "' + appKey + logParams;
      infoLog(msg);
      BugReporting.addSource(
        'AppRegistry.runApplication' + runCount++,
        () => msg,
      );
    }
    invariant(
      runnables[appKey] && runnables[appKey].run,
      `"${appKey}" has not been registered. This can happen if:\n` +
        '* Metro (the local dev server) is run from the wrong folder. ' +
        'Check if Metro is running, stop it and restart it in the current project.\n' +
        "* A module failed to load due to an error and `AppRegistry.registerComponent`
wasn't called.",
    );

    SceneTracker.setActiveScene({name: appKey});
    runnables[appKey].run(appParameters, displayMode);
  },
```

Libraries/ReactNative/renderApplication.js

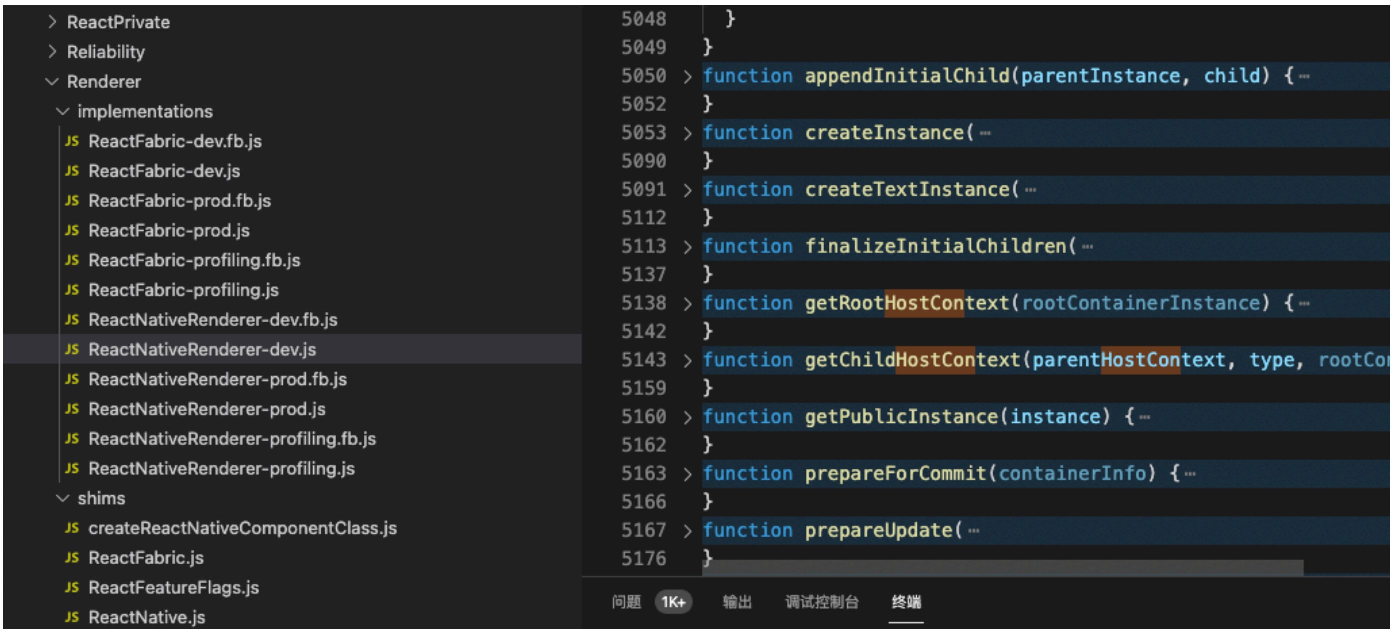
```
function renderApplication<Props: Object>(
  RootComponent: React.ComponentType<Props>,
  initialProps: Props,
  rootTag: any,
  WrapperComponent?: ?React.ComponentType<any>,
  fabric?: boolean,
  showArchitectureIndicator?: boolean,
  scopedPerformanceLogger?: IPerformanceLogger,
  isLogBox?: boolean,
  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

```
// 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



```
> ReactPrivate
> Reliability
v Renderer
  v implementations
    ReactFabric-dev.fb.js
    ReactFabric-dev.js
    ReactFabric-prod.fb.js
    ReactFabric-prod.js
    ReactFabric-profiling.fb.js
    ReactFabric-profiling.js
    ReactNativeRenderer-dev.fb.js
    ReactNativeRenderer-dev.js
    ReactNativeRenderer-prod.fb.js
    ReactNativeRenderer-prod.js
    ReactNativeRenderer-profiling.fb.js
    ReactNativeRenderer-profiling.js
  v shims
    createReactNativeComponentClass.js
    ReactFabric.js
    ReactFeatureFlags.js
    ReactNative.js
```

```
  }
}
function appendInitialChild(parentInstance, child) {⋯
}
function createInstance(⋯
}
function createTextInstance(⋯
}
function finalizeInitialChildren(⋯
}
function getRootHostContext(rootContainerInstance) {⋯
}
function getChildHostContext(parentHostContext, type, rootCo
}
function getPublicInstance(instance) {⋯
}
function prepareForCommit(containerInfo) {⋯
}
function prepareUpdate(⋯
}
```

一起实现一个render

```
import ReactReconciler from 'react-reconciler';

const rootHostContext = {};
const childHostContext = {};

const hostConfig = {
  now: Date.now,
  getRootHostContext: () => {
    return rootHostContext;
  },
  prepareForCommit: () => {},
  resetAfterCommit: () => {},
  getChildHostContext: () => {
    return childHostContext;
  },
  shouldSetTextContent: (type, props) => {
    return typeof props.children === 'string' || typeof props.children === 'number';
  },
  /**
    This is where react-reconciler wants to create an instance of UI element in terms of
the target. Since our target here is the DOM, we will create document.createElement and
type is the argument that contains the type string like div or img or h1 etc. The
initial values of domElement attributes can be set in this function from the newProps
argument
```

```
   */
  createInstance: (type, newProps, rootContainerInstance, _currentHostContext,
workInProgress) => {
    const domElement = document.createElement(type);
    Object.keys(newProps).forEach(propName => {
      const propValue = newProps[propName];
      if (propName === 'children') {
        if (typeof propValue === 'string' || typeof propValue === 'number') {
          domElement.textContent = propValue;
        }
      } else if (propName === 'onClick') {
        domElement.addEventListener('click', propValue);
      } else if (propName === 'className') {
        domElement.setAttribute('class', propValue);
      } else {
        const propValue = newProps[propName];
        domElement.setAttribute(propName, propValue);
      }
    });
    return domElement;
  },
  createTextInstance: text => {
    return document.createTextNode(text);
  },
  appendInitialChild: (parent, child) => {
    parent.appendChild(child);
  },
  appendChild(parent, child) {
    parent.appendChild(child);
  },
  finalizeInitialChildren: (domElement, type, props) => {},
  supportsMutation: true,
  appendChildToContainer: (parent, child) => {
    parent.appendChild(child);
  },
  prepareUpdate(domElement, oldProps, newProps) {
    return true;
  },
  commitUpdate(domElement, updatePayload, type, oldProps, newProps) {
    Object.keys(newProps).forEach(propName => {
      const propValue = newProps[propName];
      if (propName === 'children') {
        if (typeof propValue === 'string' || typeof propValue === 'number') {
          domElement.textContent = propValue;
        }
      } else {
        const propValue = newProps[propName];
        domElement.setAttribute(propName, propValue);
      }
    });
  },
  commitTextUpdate(textInstance, oldText, newText) {
    textInstance.text = newText;
  },
  removeChild(parentInstance, child) {
    parentInstance.removeChild(child);
  }
};
const ReactReconcilerInst = ReactReconciler(hostConfig);
export default {
  render: (reactElement, domElement, callback) => {
    console.log(arguments);
    // Create a root Container if it doesnt exist
    if (!domElement._rootContainer) {
      domElement._rootContainer = ReactReconcilerInst.createContainer(domElement,
false);
    }

    // update the root Container
    return ReactReconcilerInst.updateContainer(reactElement, domElement._rootContainer,
null, callback);
  }
};
```

其他的可参考资料：

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

问题

1. RN，你为什么选了RN?

2. RN的坑

    1. 安装、部署的坑
    2. 开发，和 React 的一些区别；`<Text></Text>`

- 路由，
- 资讯，redux, thunk,
- 点击跳转

- 图标
- 刘海屏
- RN 坑,