APKBUS

—

RN技术在腾讯课堂中的实践及优化

王华杰

目录

RN腾讯课堂中的使用情况

腾讯课堂App首页共有4个TAG，其中“首页”和“分类”两个页面是采用RN技术开发

首页

课程分类页

为什么腾讯课堂会选择RN呢？

移动端开发技术选型

H5实现

- 开发效率高
- 跨平台支持
- 可动态发布
- 体验相对差
- 调用原生能力差

终端实现

- 开发效率低
- 跨平台难
- 动态更新难
- 体验好
- 调用原生能力强

RN的特性

RN使你能够在JavaScript和React的基础上构建具有原生体验的App (Learn once, write anywhere)

- 开发效率高
- 跨平台
- 可动态发布
- 高性能&体验好

腾讯课堂是如何引入RN呢？

腾讯课堂如何引入RN

1、单业务场景

1 bundle

2、稳定性/性能欠佳

crash率高，List性能差
RN页面速度加载仍较native慢

3、基础设施支持不够

自带组件/API少，不能满足业务需求

1、开发效率有待提高，仍不够优秀
2、不适合业务大规模应用
3、不适合复杂页面应用

1、多业务、复杂业务大规模接入

多bundle

2、性能/稳定性优化

解决大部分crash，支持降级到h5
大幅提升RN页面的加载速度

3、功能增强

支持多业务打包
支持发布管理，离线包更新
定制扩展高性能组件/API

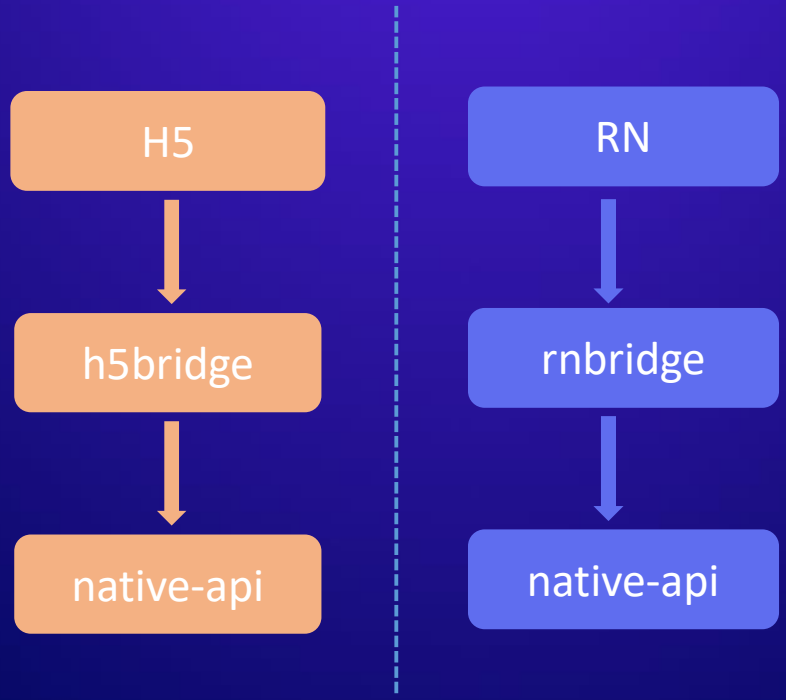
腾讯课堂RN+结构


RN+
工具
创建RN+工程
基础业务打包工具
...
Native能力
API
Storage
Account
Download
WnsTransfer
...
UI组件
RNImage
GifController
RNListView
...
发布
离线包管理
差异更新
异常降级H5
性能监控

RN与H5接口统一化

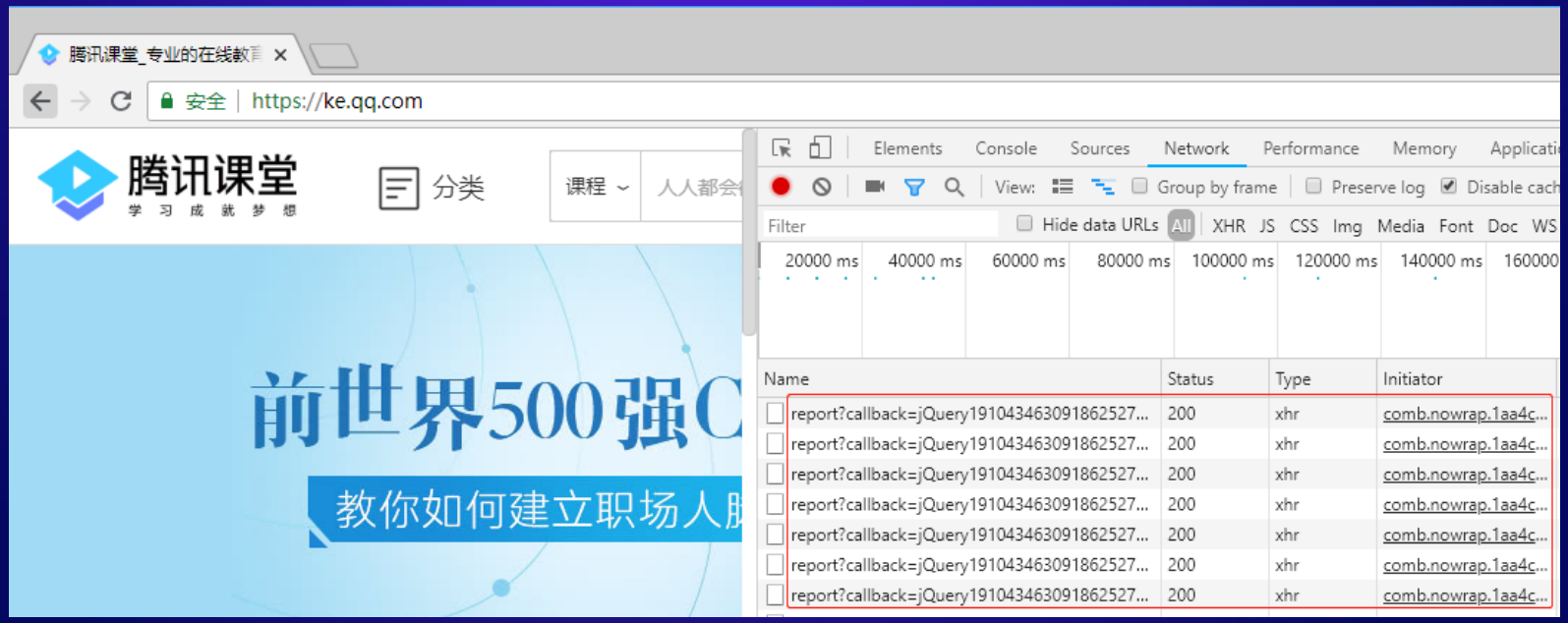
背景

原为H5提供的部分native-api现在也要为RN提供



怎么把同一份native-api同时提供给H5和RN呢?

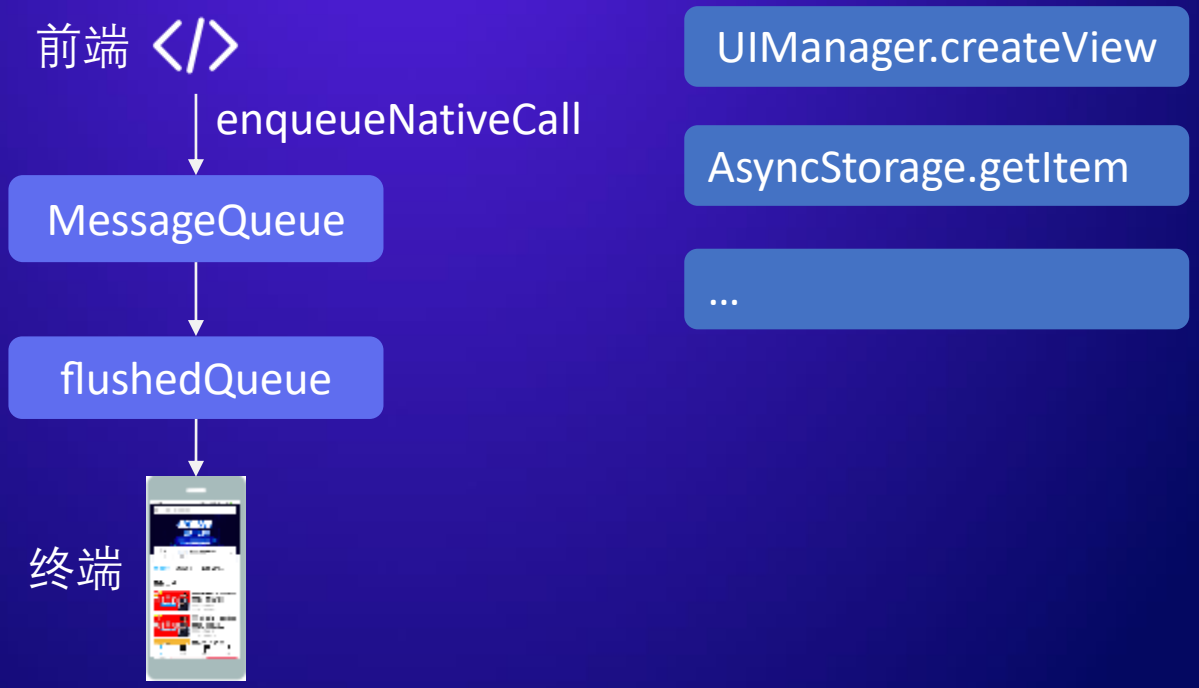
h5bridge的实现



模拟请求通信：

h5bridge://module/method/args

RN的bridge实现

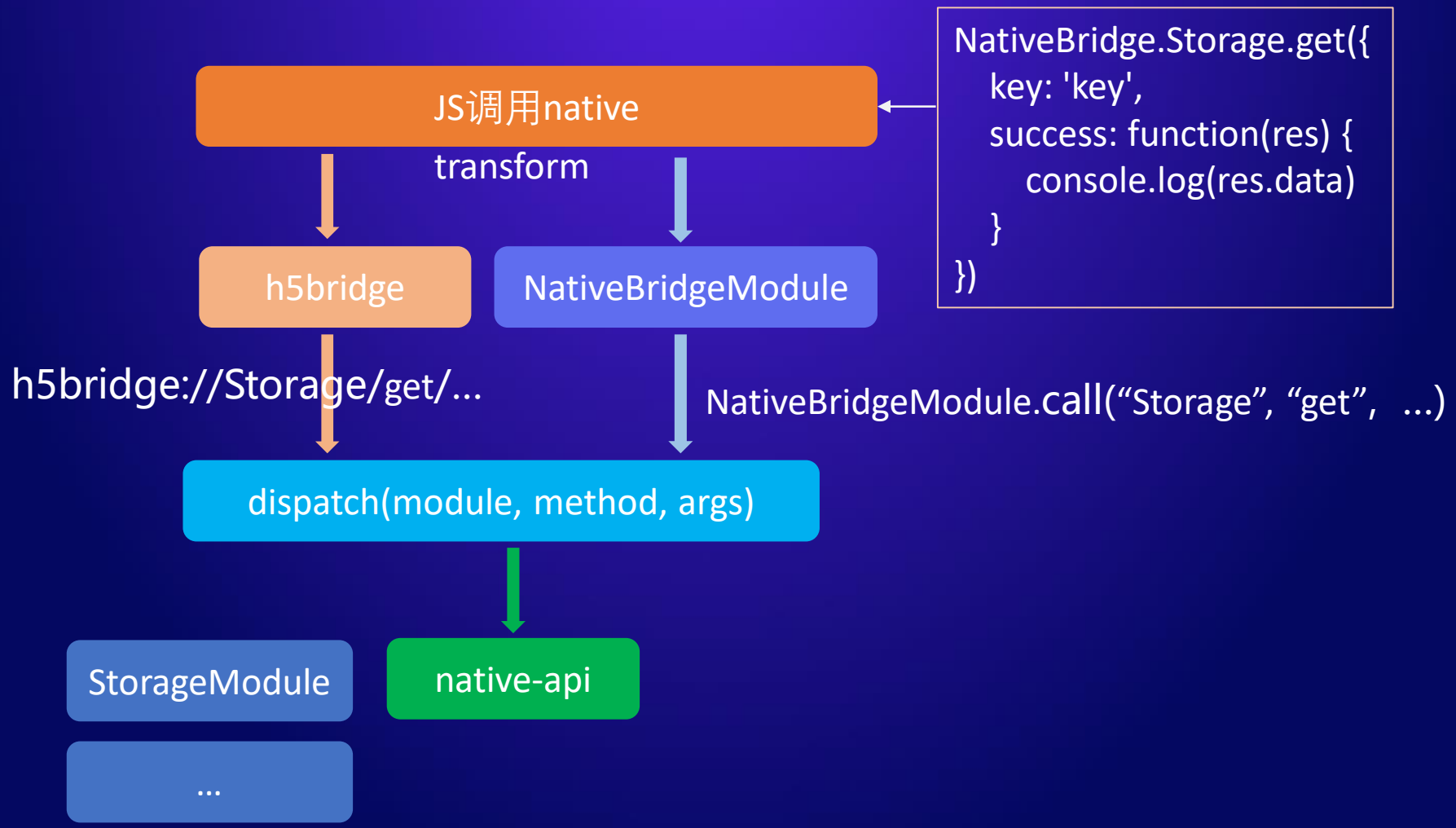


RN支持的类型：int， float, double, string, array, map, function

局限性：

- array, map中不支持function
- function仅能在参数列表的末尾，最多两个(fail, success)
- function仅能执行一次

H5与RN接口统一



参数怎么处理呢？

参数处理

```
NativeBridge.Storage.get({
    key: 'key',
    success: function(res) {
        console.log(res.data)
    }
})
```

参数转换

```
h5bridge://Storage/get/...
```

```
NativeModules.NativeBridge.call(
    “Storage”, “get”, “[...]”
)
```

参数列表转换为字符串

object中的function如何传到native呢？

```
argsList = JSON.stringify(arguments,
(key, value) => {
    if(typeof value === ‘function’) {
        _argsFunctions[_callbackId] = value
        return _callbackId++
    }
})
```

function转换为id

```
NativeModules.NativeBridge.call("Storage", "get", "[{"key": "key", "success": 1}]")
```

Native中执行function ： invokeFunction(funcId, args)

Native中module如何定义呢

- 继承ExportedModule，并把Module注册到bridge中
- 使用Exported注解导出方法
- 这个模块方法即可同时暴露给H5与RN

```
public class StorageModule extends ExportedModule {//模块导出
    public StorageModule() {
        super("Storage");
    }
    @Exported(“get”) //方法导出
    public void get(IExportedMap object) {
        String key = object.get("key");
        IFunction success = object.getFunction("success");
        success.invoke("data")
    }
}
```

H5与RN接口统一

- 解决了RN bridge局限性：
  - array, map中不支持function
  - function仅能在参数列表的末尾，最多两个(fail, success)
  - function仅能执行一次

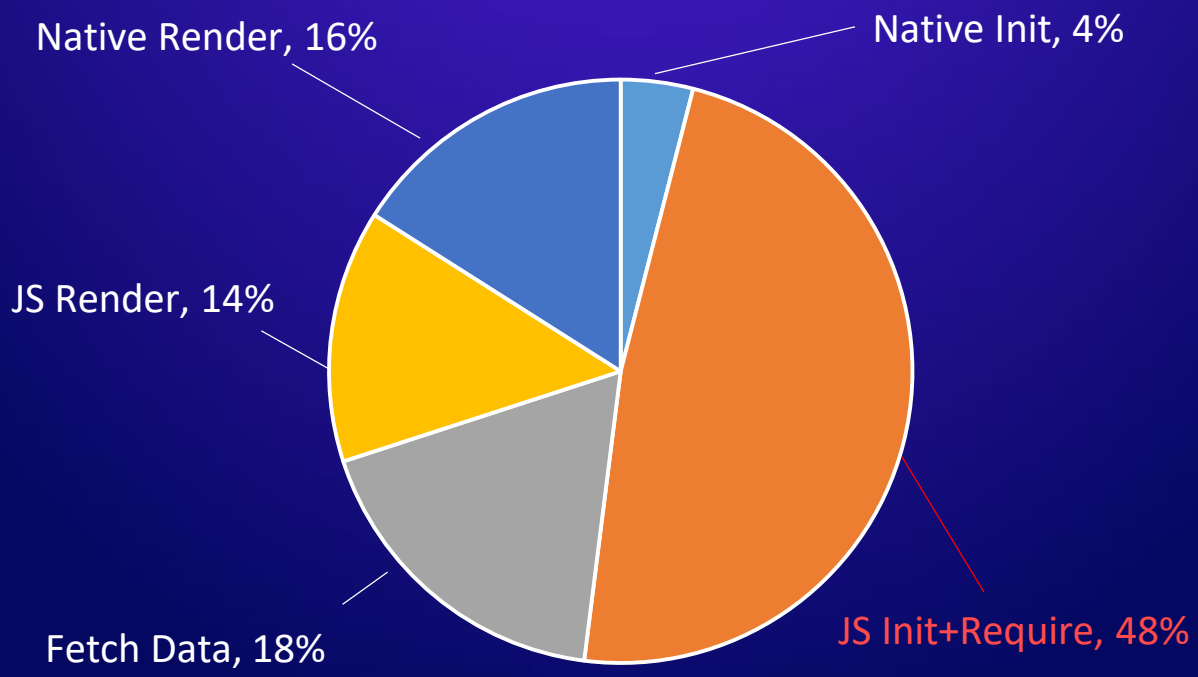
```
NativeBridge.Storage.get({
    key: 'key',
    success: function(res) {
        console.log(res.data)
    }
})
```

- 新增一个native-api更加简单，可以供RN与H5同时使用

分包与模块化加载

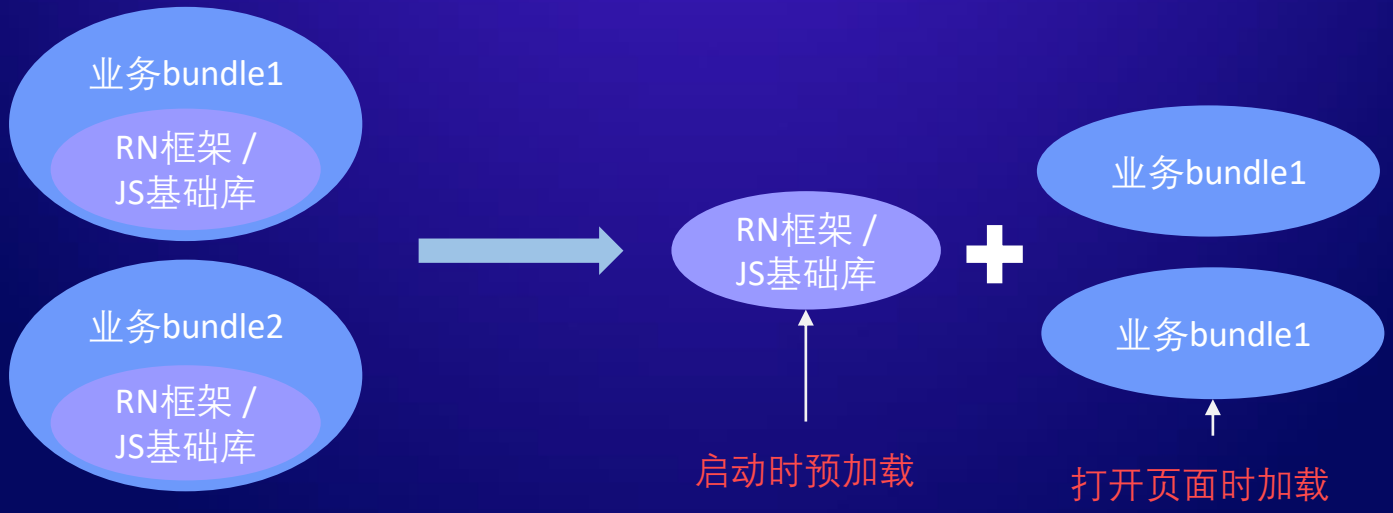
RN页面加载耗时分布

JS Init+Require -> 加载bundle耗时



拆分bundle

- RN框架大(仅RN基础库打出bundle就有500k+)
- 多业务场景下bundle冗余
- 基础bundle打包安装包里，基本不更新
- 业务bundle体积小，易动态更新



- bundle如何拆分
- 多bundle如何加载呢

RN的bundle的打包方式

bundle → 所有模块输出到一个js文件中，加载时完整加载

unbundle
- assets：每个模块是一个文本文件，打包产物有几百个文件，android的unbundle打包方式
- indexed-file：所有模块输出到一个带索引表的二进制文件，文件头中记录各个模块在文件中的相对位置，加载时按模块加载

</> 分包

一个HelloWorld Demo代码的结构如下:

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

class HelloWorld extends Component {
  render() {
    return (
      <View>
        <Text>
          Welcome to React Native!
        </Text>
      </View>
    );
  }
}

AppRegistry.registerComponent('index', () => HelloWorld);
```

头部

内容

尾部

</> 分包

使用普通bundle打包之后Bundle文件的结构如下:

```
(function (global) {

    global.__DEV__ = true;

    global.__BUNDLE_START_TIME__ = Date.now();
})(typeof global !== 'undefined' ? global : typeof self !== 'undefined' ? self : this);

__d(/* react/react.js */ function (global, require, module, exports) {
    'use strict';

    module.exports = require(13 /* ./lib/React */);
}, 12, null, "react/react.js");

  __d(/* react/lib/React.js */ function (global, require, module, exports) {
  __d(/* object-assign/index.js */ function (global, require, module, exports) {
  __d(/* react/lib/ReactChildren.js */ function (global, require, module, exports) {
  __d(/* react/lib/PooledClass.js */ function (global, require, module, exports) {
.....

require(192); ;
require(0);
```

头部：全局定义，主要是define，require等全局模块的定义

中间：模块定义，RN框架和业务的各个模块定义

尾部：引擎初始化和入口函数执行

分包方案

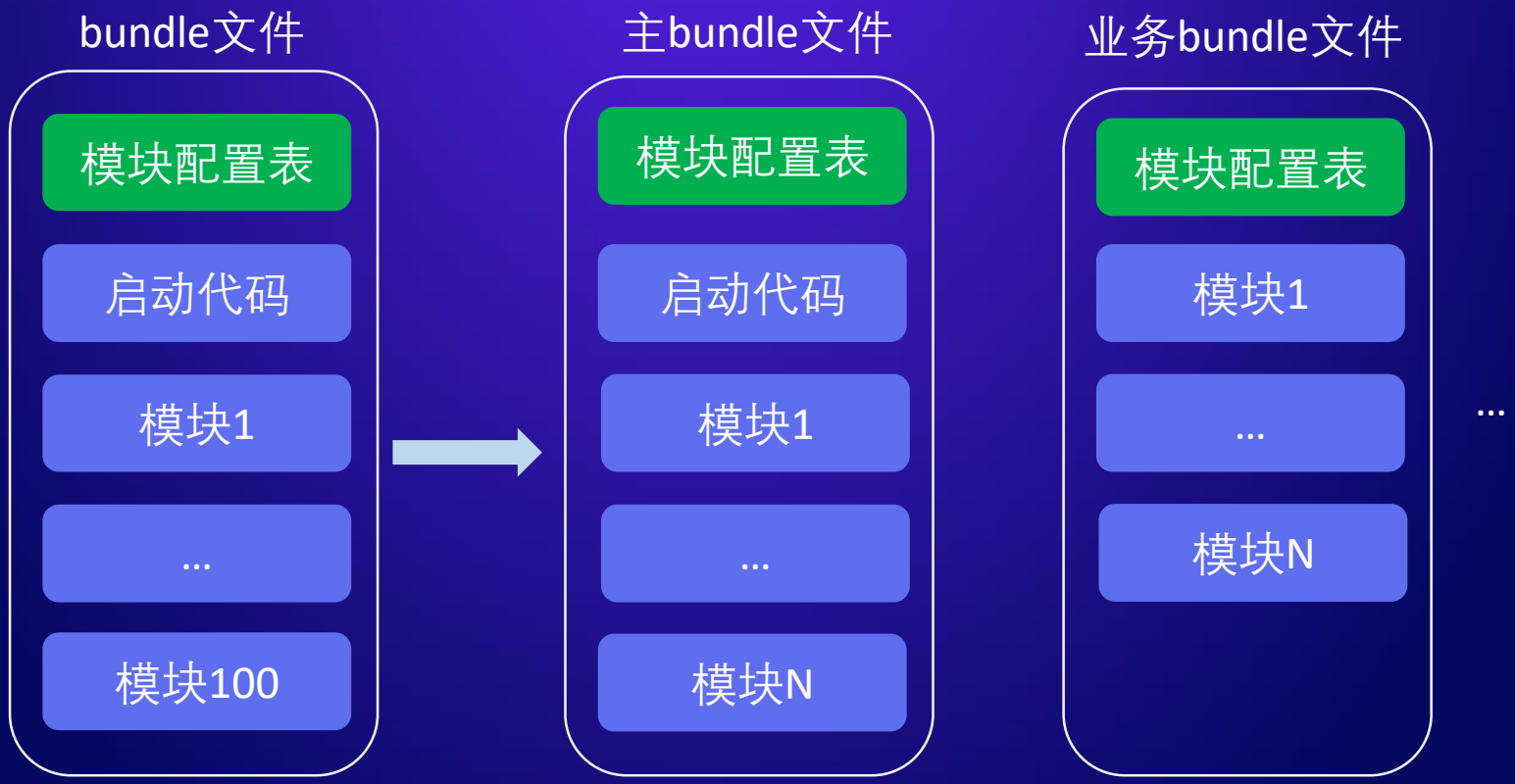
- 基于RN bundle打包方式，将bundle文件拆分成2部分(框架部分+业务模块部分)，目前主流拆包方式，
  按需加载粒度：业务包

- 基于RN unbundle assets的打包方式，将bundle文件拆分成各模块部分，可实现按需加载，
  按需加载粒度：模块

腾讯课堂RN+的分包方案是怎么样的呢?

基于RN unbundle 带索引表的打包方式，将bundle文件拆分成2部分(框架部分+业务模块部分)，可实现按需加载
按需加载粒度：模块

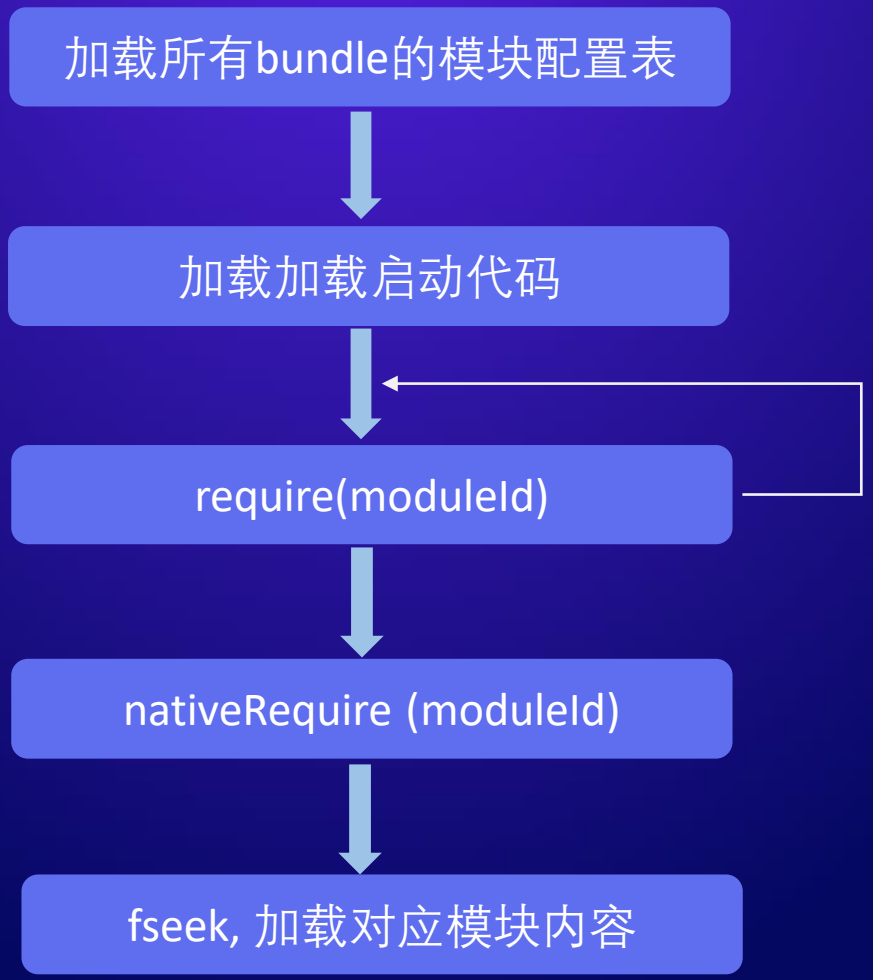
RN+的分包方案



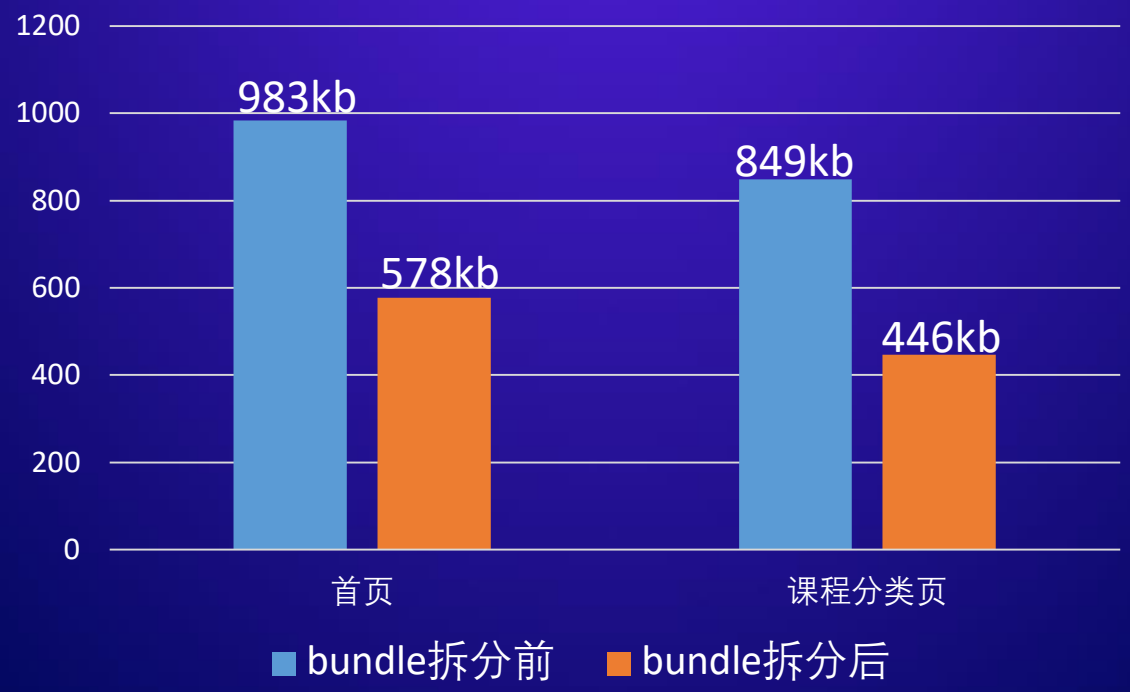

```
__d(/* react/react.js */ function (global, require, module, exports) {
    'use strict';

    module.exports = require(13 /* ./lib/React */);
}, 12, null, "react/react.js");
```

修改模块ID的生成方式，直接使用模块的路径作为其ID

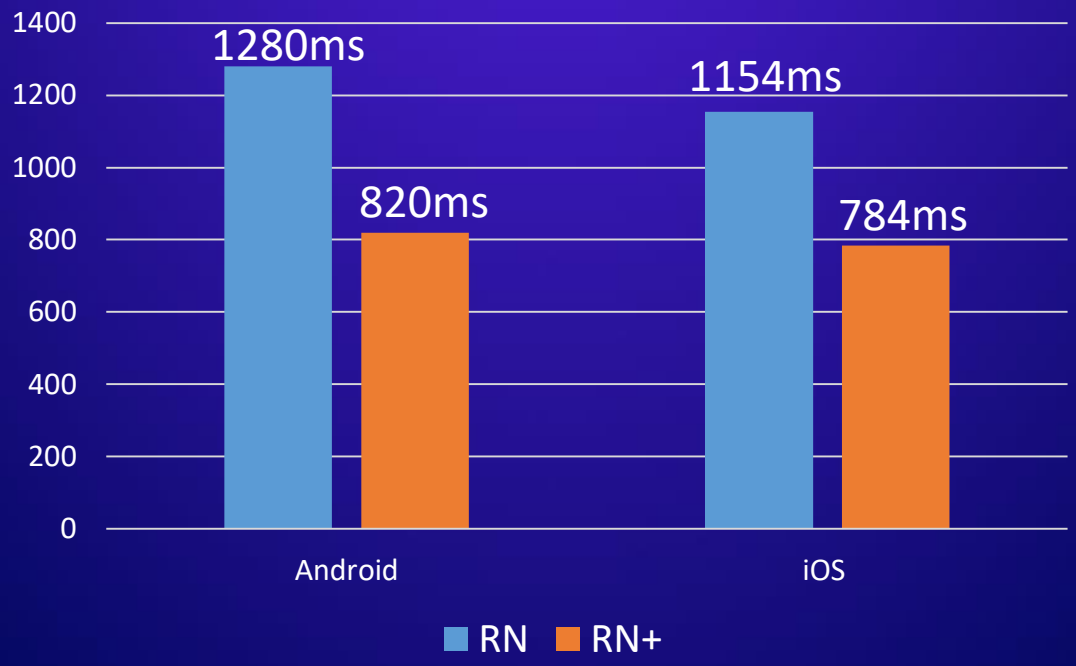
多bundle加载流程



bundle拆分效果

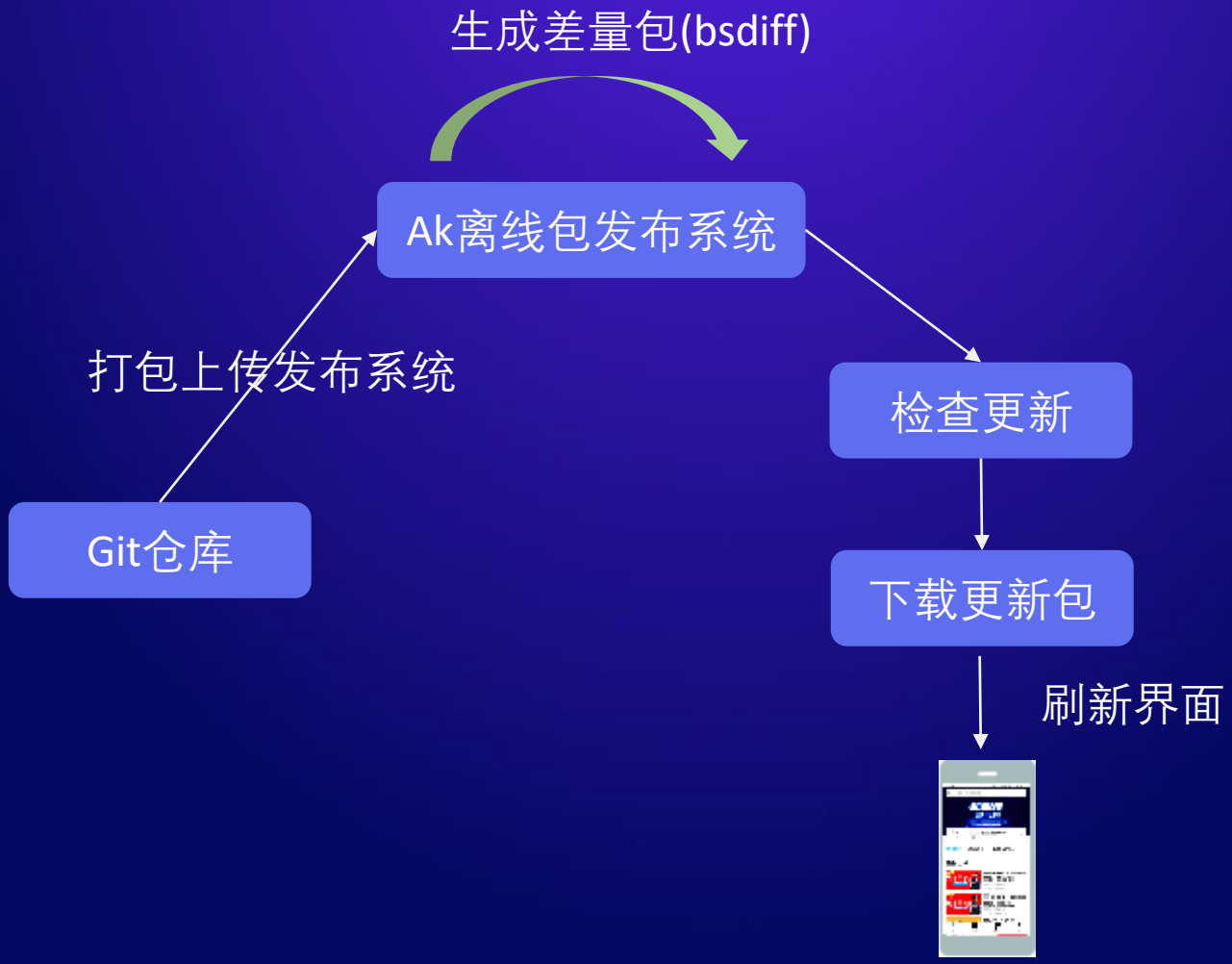

1200
1000
800
600
400
200
0
983kb
578kb
849kb
446kb
首页
课程分类页
bundle拆分前
bundle拆分后

首页加载时间

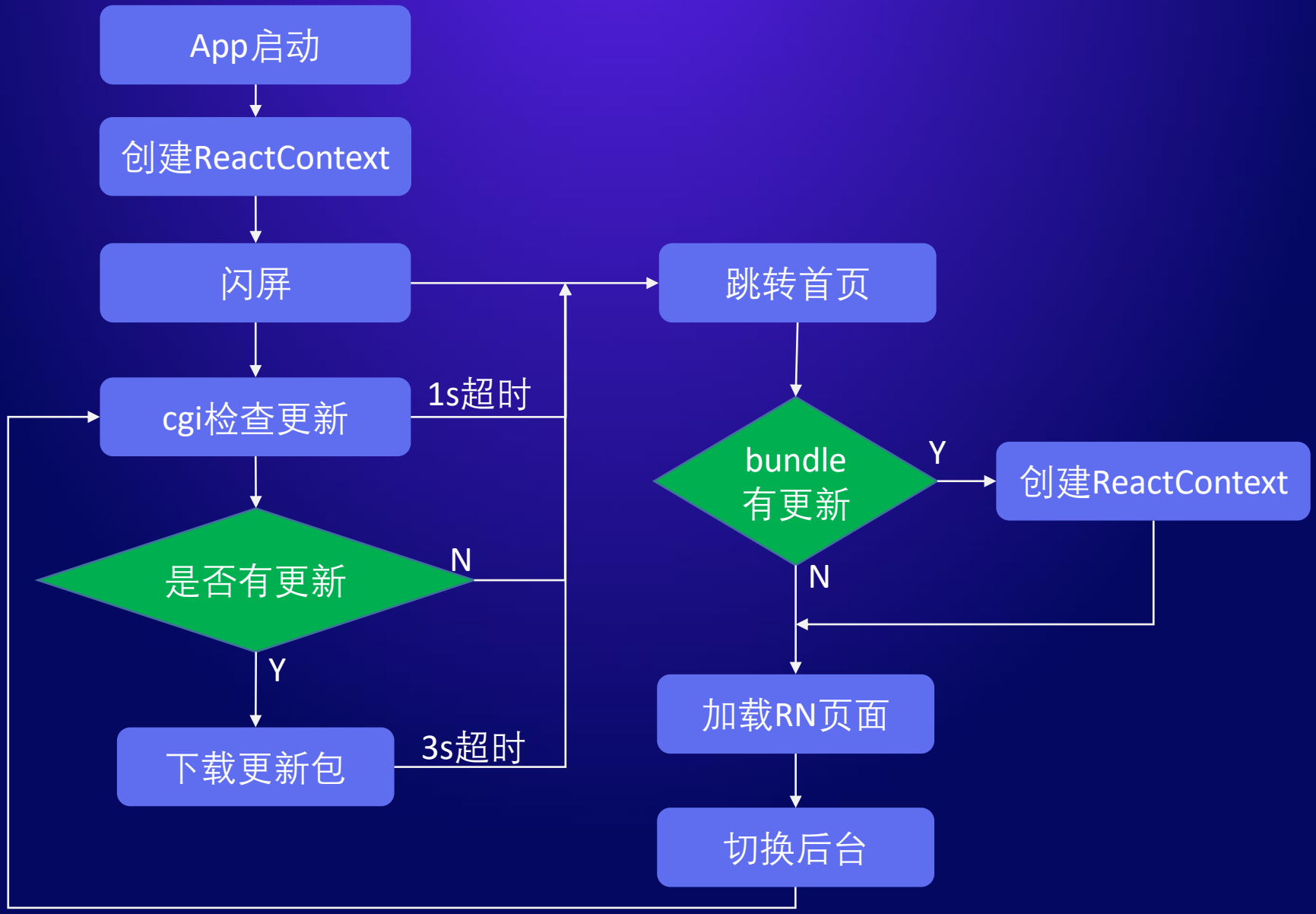

1400
1200
1000
800
600
400
200
0
1280ms
820ms
1154ms
784ms
Android
iOS
RN
RN+

发布与动态更新

bundle发布加载过程



更新bundle后如何及时刷新界面呢



异常处理与降级

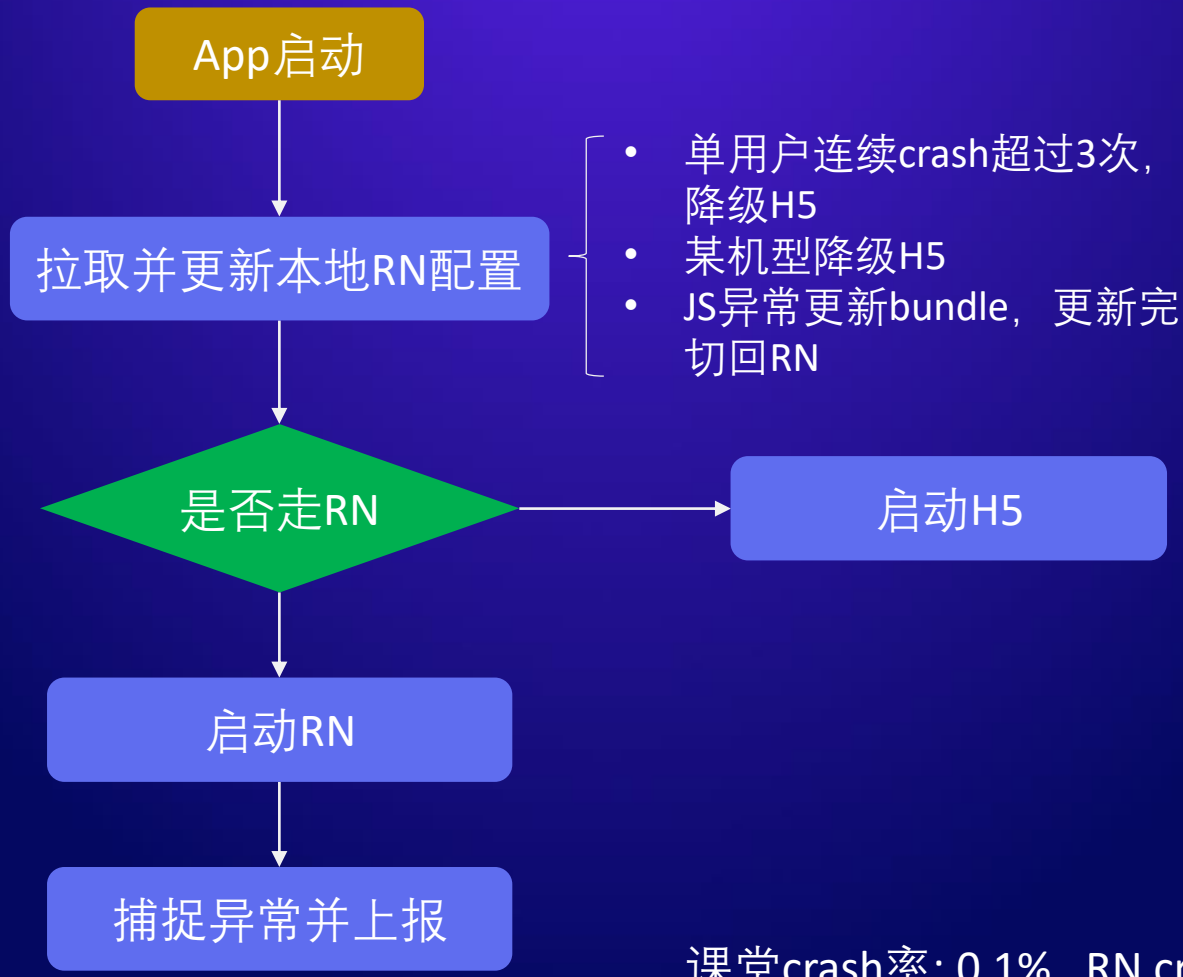
异常处理与降级

- Crash原因
  - RN源码质量，目前稳定性差
  - js代码

- 处理方式
  修复/捕捉异常+bundle版本回退或者降级H5

Crash出现的场景	Android解决方案
JSBundle 加载过程中	处理createReacrContext抛出的RuntimeException异常
业务运行过程中js报错	处理ExceptionsManagerModule抛出的JavaScriptException问题
Native模块执行报错	通过实现NativeModuleCallExceptionHandler来处理Native抛出的异常
JNI层so加载报错	try/catch 切换路径重试
ReactRootView层报错	try/catch

异常处理与降级



- 单用户连续crash超过3次，降级H5
- 某机型降级H5
- JS异常更新bundle，更新完切回RN

课堂crash率: 0.1%，RN crash率:0.001%

未来的探索

未来的探索

- 更优的性能
- 提高稳定性
- RN业务代码支持运行在浏览器上
- RN的Flutter模式

谢谢

欢迎关注安卓巴士公众号

www.apkbus.com