1.RN 监控

1.背景

ServicePoint App 目前接入的是深圳 SeaMoney-Merchant 团队提供的「中心化版本」，处于不再维护阶段，质量数据缺失比较严重，目前只有引擎加载、包下载两个阶段的监控，还有很多重要的数据指标未进行监控，对质量完全无感知，前端团队同学对质量数据有较强烈的诉求。

2.目标

- 梳理 RN 相关的技术指标
- 完善 RN 全流程的质量监控，对质量有完整的感知
    - 第一期：实现多数核心质量指标监控（页面加载过程、白屏相关监控）
    - 第二期
        - 核心质量指标：JS 错误率
        - 体验指标：FPS（JS、Native）

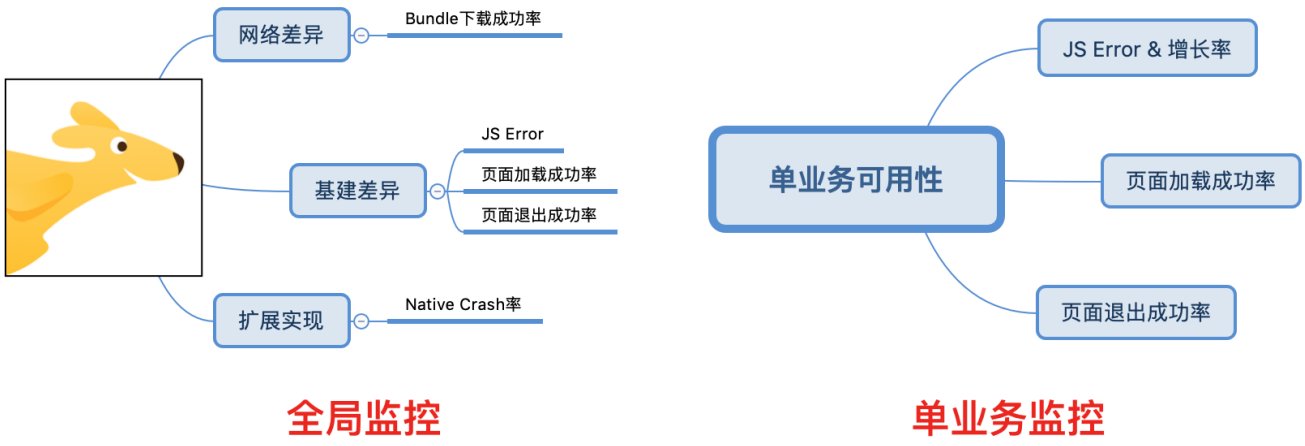
3.方案

3.1 监控方案调研

这里主要是调研国内大规模使用 RN 的公司：携程、美团。携程开放出来的资料关于监控的比较少，RN 框架的架构改造和优化介绍的比较多，不过还是能看出一些思路，比如携程统计了

- 首屏加载性能

而美团统计的比较多，可以看得出



我在美团使用 React Native 框架的过程中，框架本身做了非常详细的指标统计，是值得我们去参考的。

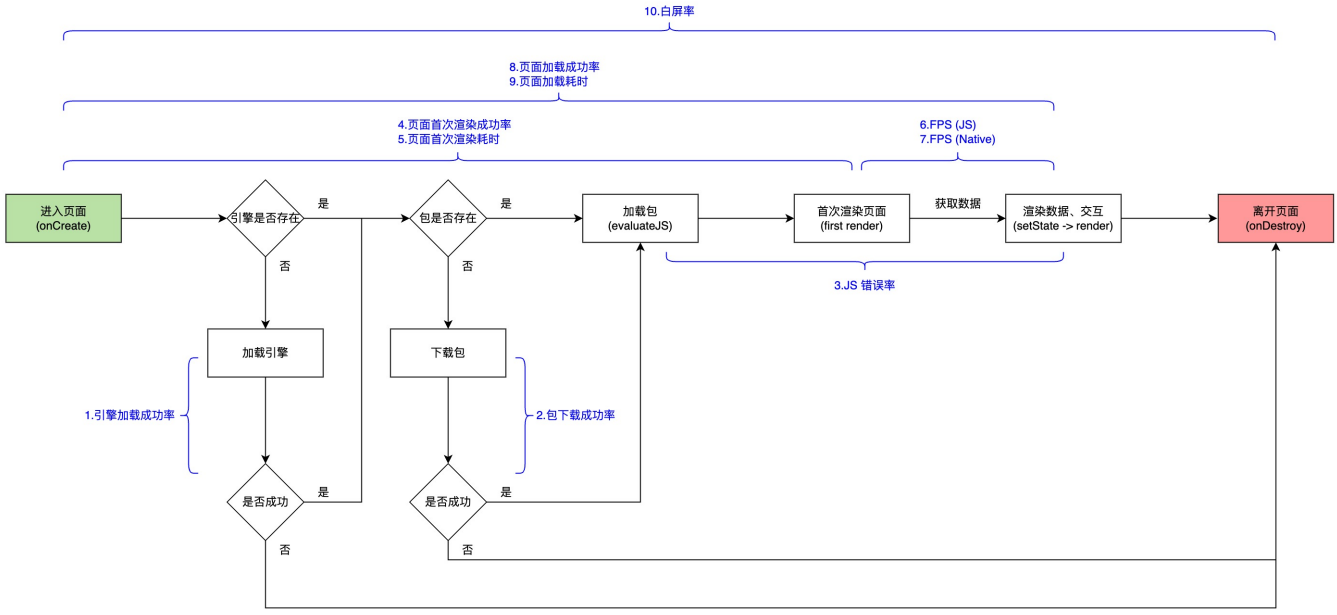
公司的 RN 团队也在指标建设方面也开始逐步发力，完整的建设情况：RN端监控指标，目前 Partner App 也在做一些类似的监控：增强Partner App的监控能力，下面的监控方案中会和他们有一定的区别，目前未找到白屏监控的相关方案。

关于共建的考量

- 短期：业务可自己快速得到的指标，先自主实现，共建的话周期会比较长（尤其目前我们接入的是中心化版本，RN 团队不再维护，不可能配合我们去加监控）
- 长期：和 RN 团队共建监控体系

3.2 监控方案设计

3.2.1 监控全流程指标



这里我们只挑选 5 个还没监控的指标来做

- 页面首次渲染成功率
- 页面首次渲染耗时
- 页面加载成功率

- 页面加载耗时
- 白屏率

序号	指标	定义	是否已实现	该要说明
1	引擎加载成功率		是 框架内部实现	
2	包下载成功率		是 框架内部实现	
3	JS 错误率		否	
4	页面首次渲染成功率	页面「首次 render」次数 / 进入页面总次数	否	• 可以在 Component 的 constructor 函数尾部
5	页面首次渲染耗时	从进入页面到「首次 render」的时间	否	
6	FPS(JS)		否	
7	FPS(Native)		否	
8	页面加载成功率	页面「首次数据请求」结束后的「 render 成功」次数 / 进入页面总次数	否	• 可以在 Component 的 componentDidUpdate 中加（检测有数据返回后设置标识位）
9	页面加载耗时	从进入页面到「首次数据请求」结束后的「 render 成功」时间	否	
10	白屏率	从进入页面（未退出页面）> 4s 时页面没有任何绘制的次数 / 进入页面总次数	否	白屏的判断依据（叶节点渲染区域 + 视觉） • 节点渲染区域：ReactRootView 节点渲染区域在屏幕内 • 视觉判断法：截图判断图片像素均为背景色（或白色）

3.2.2 页面首次渲染监控

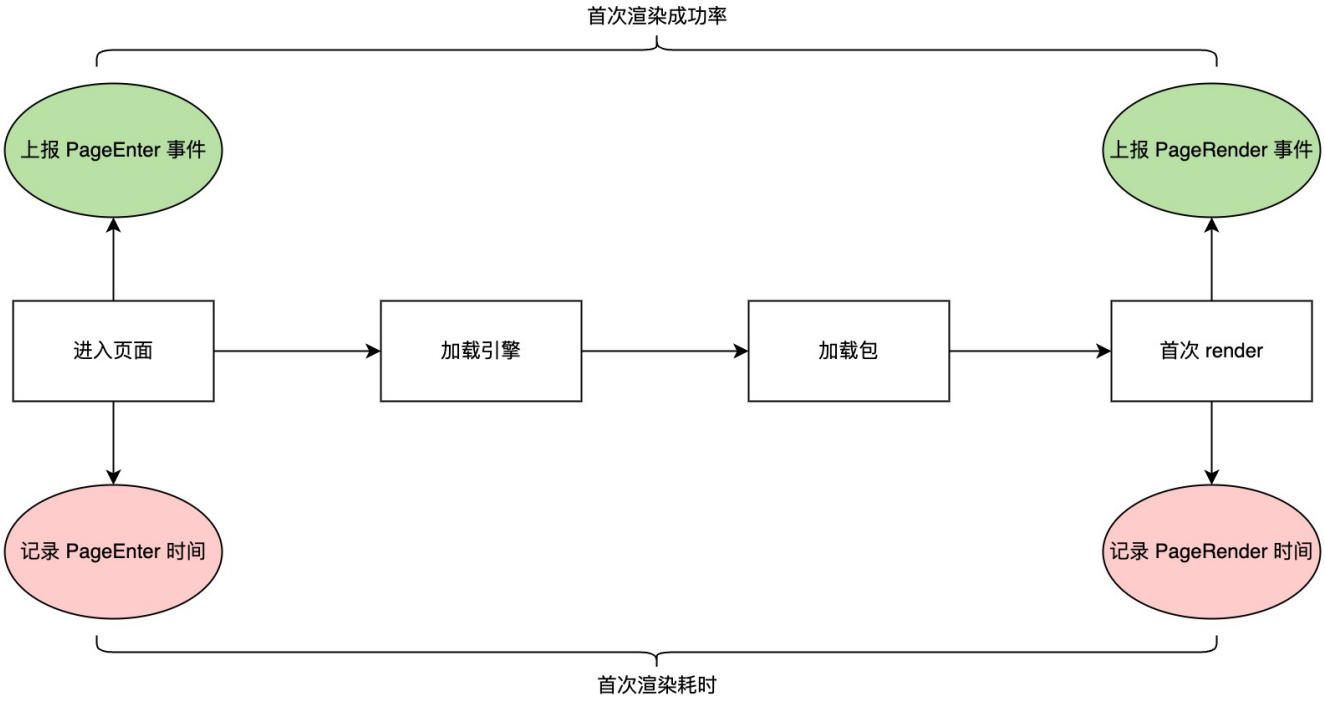
指标定义

- 页面首次渲染成功率：页面「首次 render」次数 / 进入页面总次数
- 页面首次渲染耗时：从进入页面到「首次 render」的时间

统计原理：用户进入页面，上报一个「进入页面」事件，并记录进入时间；**页面 render 函数第一次执行**，上报一个「首次 render」事件，并记录时间。两个事件 count 相除得到成功率，两个事件时间的差得到耗时。

这里只统计首次 render

- 可以在 Component 的 constructor 函数尾部
- 也可以在 Component 的 ComponentDidMount 中添加



方案对比

- **公司方案：**未统计
- **业内方案：**和美团差不多

3.2.3 页面数据渲染监控

指标定义

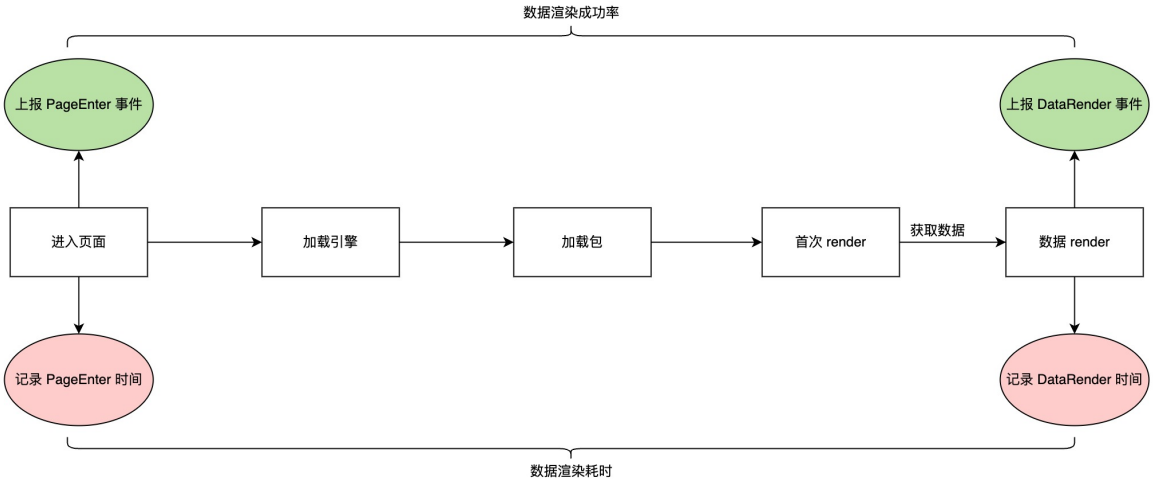
- 页面加载成功率：页面「首次数据请求」结束后的「 render 成功」次数 / 进入页面总次数
- 页面加载耗时：从进入页面到「首次数据请求」结束后的「 render 成功」时间

统计方案 A：用户进入页面，上报一个「进入页面」事件，并记录进入时间；**页面主数据接口返回后，render 函数第一次执行完成**，上报一个「数据 render」事件，并记录时间。两个事件 count 相除得到成功率，两个事件时间的差得到耗时。

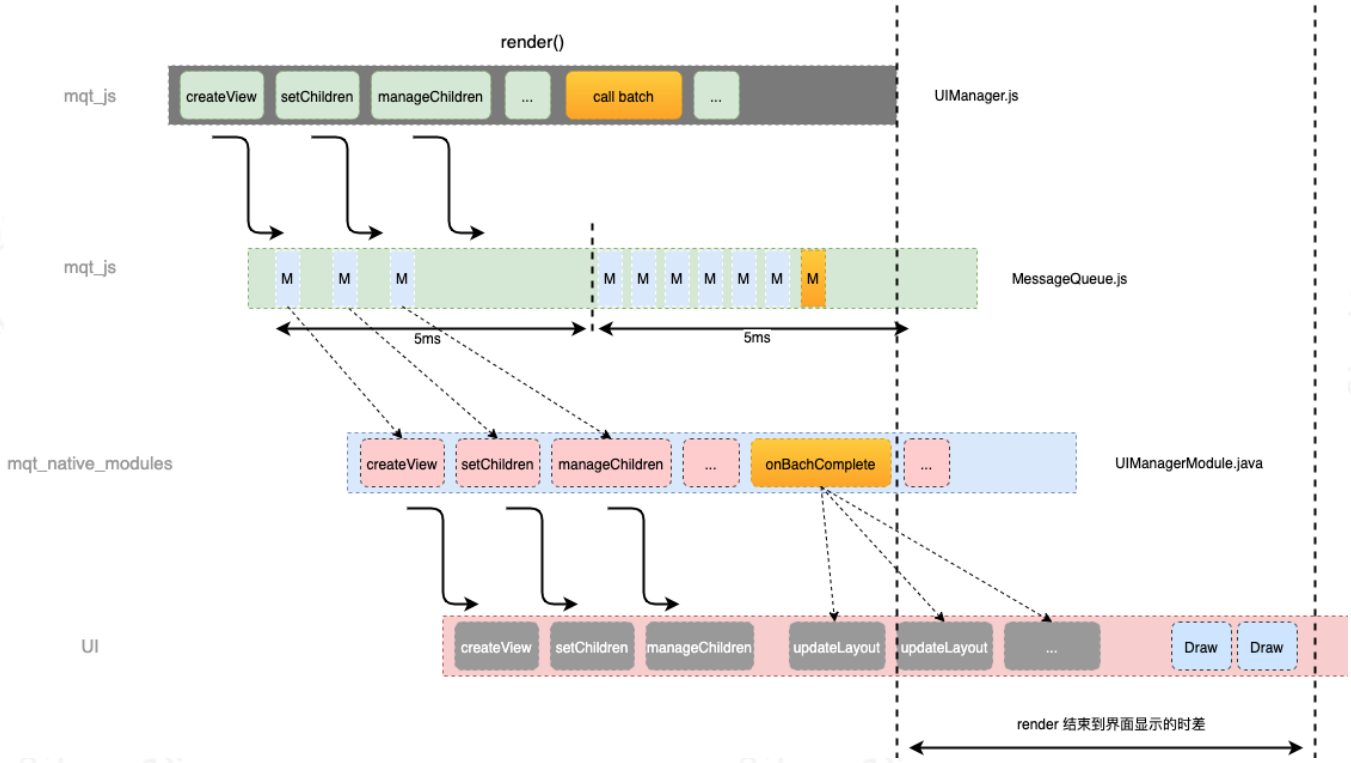
这里的统计需要满足两个条件

- 主数据接口返回数据
- render 函数执行完毕

可以在数据接口返回后置一个标识位，在 Component 的 ComponentDidUpdate 中检测到标识位则上报，上报过一次后，不再重复上报

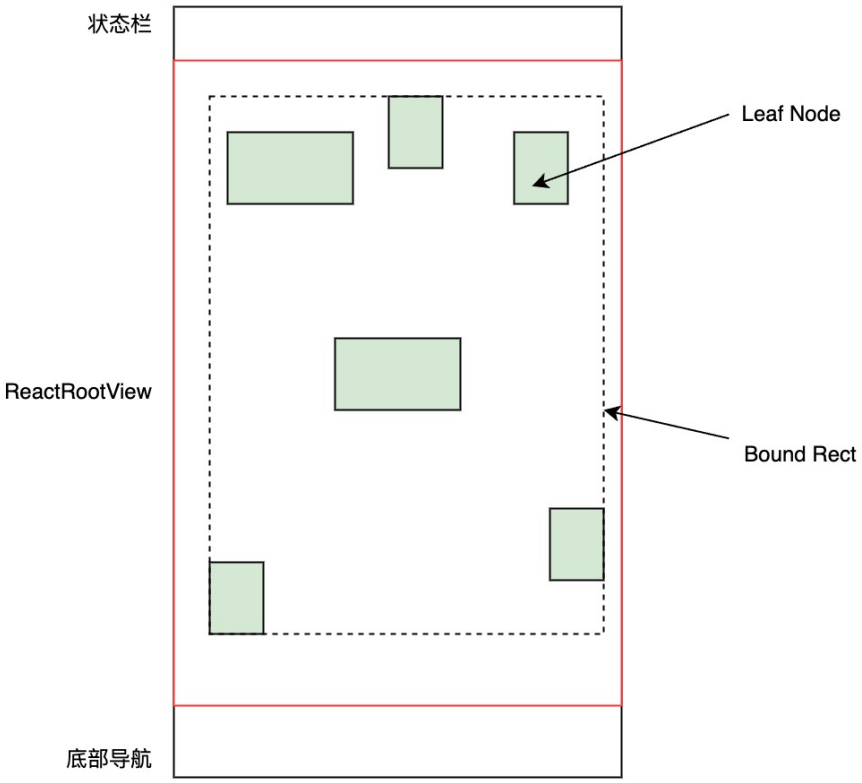


统计方案 B：统计方案 A 只是统计了 render 函数执行完毕的时间，并不代表元素真正绘制到屏幕上，中间还有一小段时间的差距，渲染原理图如下（选自我以前整理的技术文章：深入探索 React Native 技术原理）

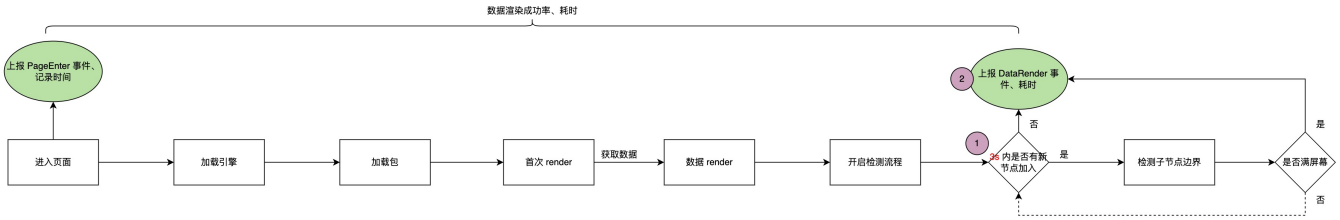


提升方案是在收到数据 render 之后，监听 Native View 的节点变化，检测叶子节点的绘制区域在屏幕中的位置

- 如果所有子节点的绘制区域已经充满整个屏幕，则上报数据
- 如果未充满屏幕，等待 3s，未有新节点变化，则上报最后一次节点更新时间的数据



完整的方案如下



1. 3s 的这个目前业内没有可参考的数据，只能根据实际情况去做调整
2. 上报的是最后一次节点添加时间，排除 3s 后的
3. 这里可能存在极值，比如持续在 3s 内子节点不断变化，且屏幕未充满

几个方案对比（目前选统计方案 A）

- **公司方案：**检测页面完成
- **统计方案 A：**
  - 优点：成本最低，数据比较准
  - 不足：比用户真实看到的绘制时间偏小点
- **统计方案 B**
  - 优点：比较接近用户真实看到的加载时间
  - 不足：成本高，会有一些极值出现，需要去调整

3.2.4 白屏检测方案

白屏检测业内主要是两个大的方向

- 视觉法：截屏看是否是白色
- 节点监控：节点是否渲染在可视区域

目前能看到的方案大多是基于 WebView、纯浏览器的，RN 的白屏监控暂时没有相关的技术实现。

在开始检测的时机点上，业内也暂时无一个通用的标准

- 百度小程序：6s
- 公司方案：5s（白屏开始检测时间）

这里我们定 4s，依据是谷歌对 LCP 的一个评估，4s 后还未加载成功则是最差的级别：https://web.dev/lcp/

指标定义：从进入页面（未退出页面）> 4s 时页面没有任何绘制的次数 / 进入页面总次数

统计方案：根据 ReactRootView 子节点渲染区域是否在屏幕内去做判断：RN 页面节点对应关系，这里

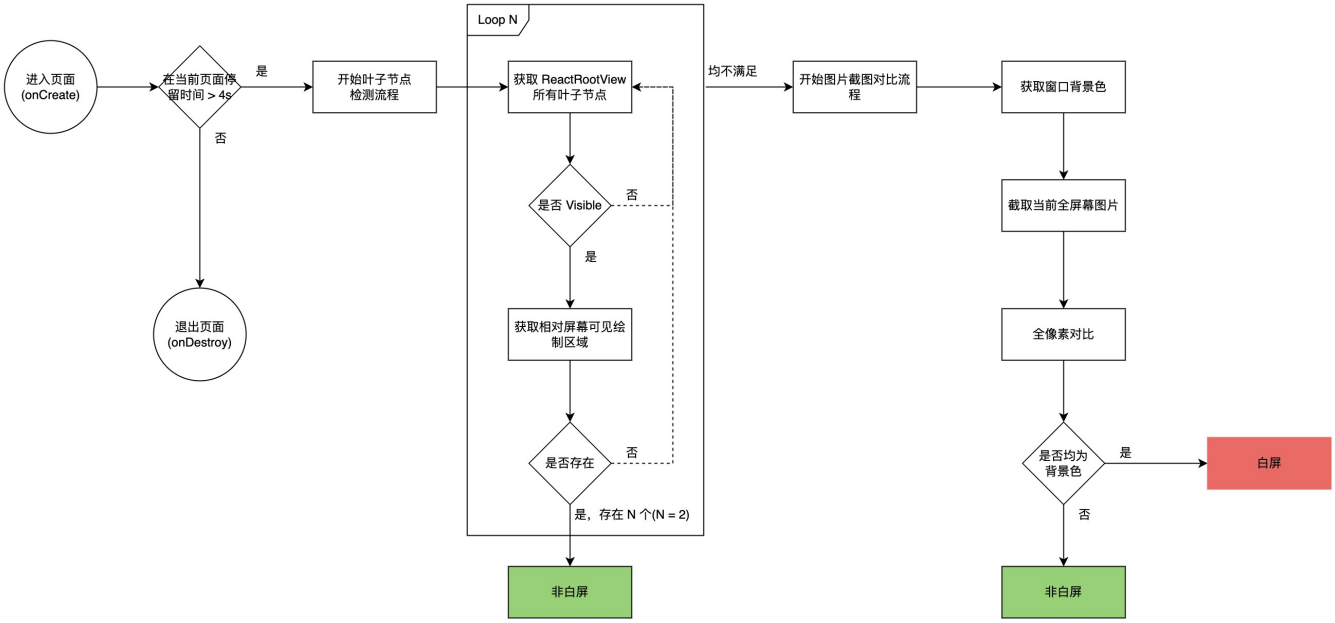
1. 检测所有子节点
    a. 漏判的情况：全屏幕的容器节点必然满足条件，会被判定为非白屏幕，但是如果叶子节点渲染出错，无叶子节点渲染到屏幕上，则会造成漏报
    b. 误判的情况：不存在
2. 只监控叶子节点：
    a. 漏判的情况：叶子节点在可视区，但是实际上本身白屏，如 ReactTextView、ReactImageView 等无任何绘制
        i. 无异常发生：考虑到正常业务场景中，存在这种情况的可能性为 0
        ii. 有异常发生：一般来说，节点创建成功，props 也会设置成功，所以不绘制的概率较低，可以多判断几个节点（> 2，这里做了个配置，可改），进一步降低概率

```
@ReactMethod
public void createView(int tag, String className, int rootViewTag, ReadableMap props) {
  if (DEBUG) {
    String message =
        "(UIManager.createView) tag: " + tag + ", class: " + className + ", props: " + props;
    FLog.d(ReactConstants.TAG, message);
    PrinterHolder.getPrinter().logMessage(ReactDebugOverlayTags.UI_MANAGER, message);
  }
  mUIImplementation.createView(tag, className, rootViewTag, props);
}
```

    b. 误判的情况：对于一些极端页面，无叶节点的（继承自 View 的），则统计不到子节点绘制区，会被判定为白屏，造成误报
        i. 对于误报的，可以通过截图识别方式去排除

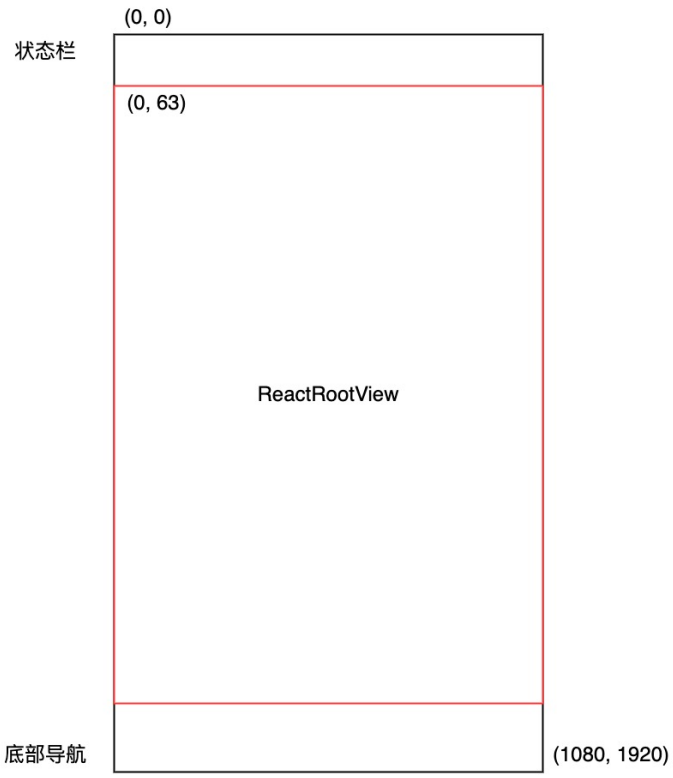
选第 2 种实现方式，先比较节点区域，性能损失较小，过滤掉大部分的情况，对于满足白屏的，再使用截图识别，这样对用户的设备资源消耗最小，这里做了个线上统计，如果性能影响较大，可通过调整子节点 Count（目前是 2）来控制

- 图片对比率：走到图片对比逻辑的次数 / 进入页面总次数
- 图片对比耗时：分析线上用户图片对比的时长



说明

1. 这里截图是截取整个屏幕 DecorView 的图片，不能截取 ReactRootView 的图片，截取 ReactRootView 不是视觉上看到的图片
2. 比较区域必须是 ReactRootView 所在屏幕的区域
3. 需要和背景色来比，而不是和白色比，如果没有背景色，才使用白色



线下模拟验证：白屏检测线下测试验证

白屏个案上报：初期为了验证白屏检测的准确性，可以把白屏的图片上传到服务器

- 业务图片服务：目前不可用
- Firebase 文件服务：调研中

3.3.数据指标设计与统计

3.3.1 数据协议

总体使用 ShopeeTracker SDK 上报，已经有公共字段

类型	字段	含义	说明
公共字段	country	国家	
	app_id	应用 ID	
	app_version	应用版本好	
	platform	平台	Android / iOS
	os	操作系统	
	os_version	操作系统版本	
	brand	品牌	
	model	型号	
	deviceid	设备唯一标识	
	userid	用户 ID	
	event_id	事件 ID	

	event_timestamp	事件发生时间	
	network_type	网络类型	

以下是补充业务字段

类型	字段	含义	说明
业务公共字段	shop_id	门店 ID • 默认值：0	
	engine_state	引擎状态 • 默认值：0 • 新加载：1 • 已预加载：2	
	bundle_state	包状态 • 默认值：0 • 未下载：1 • 已下载、未预加载：2 • 已预加载：3	
	bundle_name	包名称	
	bundle_version	包版本	
	page_name	页面名称	
事件字段	event_name	事件名	如：白屏率（blank_rate）
	value	事件值	• 0：失败 • 1：成功
	duration	事件耗时	耗时：1000ms
	duration_tag	耗时分布	• 0.0 ~ 0.5 • 0.5 ~ 1.0 • 1.0 ~ 1.5 • ...

3.3.2 事件名

- 页面进入（enter）
  - 上报 PV：**page_enter_count**
- 页面首次渲染（render）
  - 上报成功率：**page_first_render_count，value 为 1**
  - 上报时长：**page_first_render_duration**
- 页面数据渲染（ready）
  - 上报成功率：**page_data_render_count，value 为 1**
  - 上报时长：**page_data_render_duration**
- 白屏率：在当前页面停留 > 4s，检测到白屏
  - 上报白屏率：**page_blank_count，value 值为 1**
  - 上报时长：**page_blank_duration**
- 页面离开（leave）
  - 上报成功率（如果未上报过）：**page_first_render_count，value 为 0**
  - 上报成功率（如果未上报过）：**page_data_render_count，value 为 0**
  - 上报成功率（如果未上报过）：**page_blank_count，value 为 0**

序号	指标	事件名	说明
1	页面首次渲染成功率-server	`count(page_first_render_duration) / count(page_enter_count)`	服务端计算
	页面首次渲染成功率-client	average(page_first_render_count)	客户端计算
2	页面首次渲染耗时	page_first_render_duration	客户端计算（render - enter）

3	页面加载成功率-server	count(page_data_render_duration) / count(page_enter_count)	服务端计算
	页面加载成功率-client	average(page_data_render_count)	客户端计算
4	页面加载耗时	page_data_render_duration	客户端计算（ready - enter）
5	白屏率-server	count(page_blank_duration) / count(page_enter_count)	服务端计算
	白屏率-client	average(page_blank_count)	客户端计算

4.推进进展

时间线	事项	进度	数据准确性验证	其他说明
2022-08-25 ~ 2022-09-06	1. 整理初步技术方案 2. 实现页面首次渲染指标监控 3. 实现页面加载指标监控 4. 白屏监控 5. 数据报表配置	90%		
2022-09-07 ~ 2022-09-20	1. 按背景、目标、调研、方案、推进等几个重点部分去描述，重点补充方案调研和方案设计细节部分 2. 页面加载指标监控方案完善，补充可选方案 3. 白屏监控方案调整和实现 a. 由「数子节点数量」变成判断叶子节点绘制区域 b. 图片对比变成全像素对比	100% 09-21 live	白屏检测线下测试验证	
2022-09-21 ~ 2022-10-09	1. 首次渲染成功率、数据渲染成功率增加客户端计算的方式 (09-29 live) 2. 整理数据统计结果 a. 白屏率：数据分析-白屏率 b. 首次渲染耗时：首次渲染耗时分布 c. 数据渲染耗时：数据渲染耗时分布 d. 首次渲染成功率：首次渲染成功率-server e. 数据渲染成功率：数据渲染成功率-server 3. 白屏率增加客户端计算的统计方式 (10-14 live) 4. 整理剩余的数据指标，主要是推进 MDAP 修改 bug a. 首次渲染成功率 b. 数据渲染成功率		• 白屏率 • 09.26 ~ 09.27：Leaf = 2 • 09.28 ~ 09.29：Leaf = 1 • 09.30 ~ 10.02：Leaf = 3 • 10.03 ~ 10.04：Leaf = 5 • 10.04 ~ ：Leaf = 1 • 耗时分布：26号上午修复的分位，开始统计数据 • 成功率：统计一周（0926 ~ 1002）	目前看到数据指标不是很准，可能存在的原因，大概有 0.1% 的误差 • 两个事件上报成功率的不同 • 线下数据干扰 所以，单独加了一个客户端统计成功率的口径，需要等 MDAP 平台修改 bug 才能看
2022-10-10 ~	1. MDAP 上重新配置了计算成功率的 client 计算方式：sum / count 2. 首次渲染成功率统计：首次渲染成功率-client 3. 数据渲染成功率统计：数据渲染成功率-client			

5.成果

5.1 指标查询与报表分析

数据查询平台

- 页面加载：https://mdap.shopee.io/boussole/ssc-spx-servicepoint-dashboard/dashboard/view/cYuvMUHBRVPQE8el?mode=preview
- 白屏：https://mdap.shopee.io/boussole/ssc-spx-servicepoint-dashboard/dashboard/view/kfN5P8aVjCqhloLV?mode=preview

数据指标基本验证完毕

- 首次渲染耗时：首次渲染耗时分布
- 首次渲染成功率：首次渲染成功率
- 数据渲染耗时：数据渲染耗时分布
- 数据渲染成功率：数据渲染成功率
- 白屏率：页面白屏率

对于以下两个指标，通过服务端计算方式得到的，大概有 0.1% 左右的误差（可以从 首次渲染成功率一周数据 和 预请求-数据分析 两个数据佐证），后续不再使用

- 首次渲染成功率：首次渲染成功率-server
- 数据渲染成功率：数据渲染成功率-server

5.2 结论

1. 目前「四个扫码页面」秒开率 47.4% 左右，可以进一步提高，参考美团可以做到 90% 的目标
2. 白屏率基本在 **0.2%** 左右（*这个数据由于 MDAP 上报丢数据的情况大概会有 0.1% 的误差，所以大概白屏率在 01.% ~ 0.3% 之间*），RN 的白屏率这块没太多业内标准可参考，不过可以参考 Native 的 Crash 率标准来定（大概率是 JS Crash 导致），DAU 小于 1w 的产品，按 0.1% 为标准（*不过毕竟和整个 App 崩溃相比，严重程度会小一些，可以酌情降低一点目标*）

6.参考文献

Moles：携程基于 React Native 的跨平台开发框架

携程开源RN开发框架CRN

React Native在美团外卖客户端的实践

H5 白屏检测方案实践

百度小程序白屏优化

如何利用图片对比算法处理白屏检测