数据预请求方案

1.背景

数据预加载是业内常用的提升页面加载速度的手段之一。对于 Native 页面，跳转过程本身并不是很耗时，但是页面打开后请求网络是需要等待的，所以提升的手段就是在当前页预测下一个可能打开的页面，提前请求下一个页面的数据，这样当用户点击进入下一页，即可展示，不用等待网络请求返回，可以极大提高页面加载速度。

对于 RN / Flutter 之类的页面，在页面跳转的时刻还需要执行「加载引擎」、「加载 Bundle 包」之类的动作，之后才会真正执行页面的逻辑，去请求数据，如果是首次加载，则会比较耗时，这部分完全可以利用来做数据的预请求。

2.目标

实现数据预加载框架，减少 RN 页面首次打开时间

- 首次打开：利用加载引擎、加载包的时间来提前发出请求
- 二次打开（无引擎加载、包加载）：对于执行较慢的低端机，提前发出请求，等业务真正执行到网络请求 JS 的时候

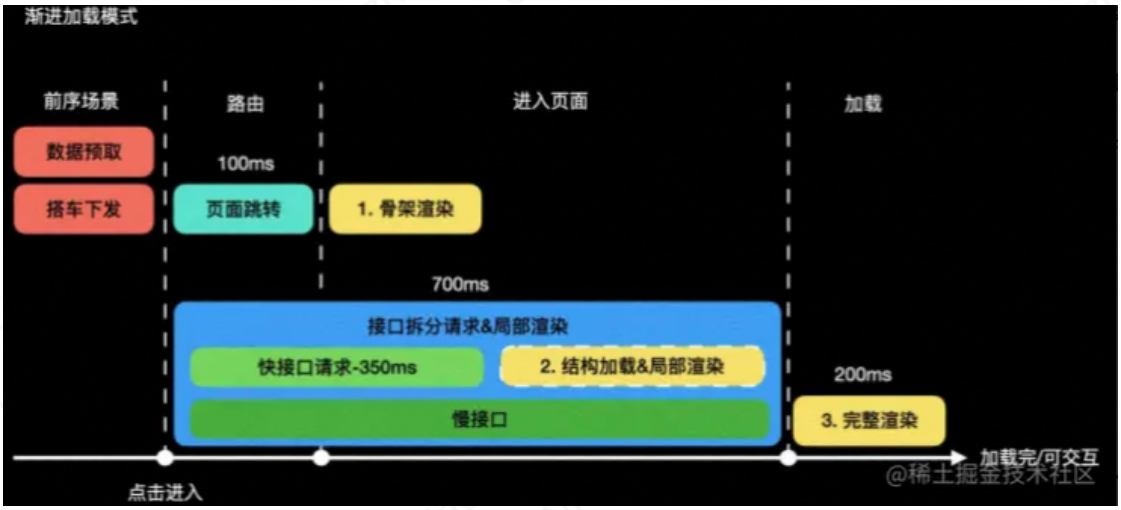
3.方案

3.1 方案调研

虽然大厂都有做类似的优化，不过在技术方面方面以及数据方面并没有开放出来技术文章

- 美团在提升秒开率方面：引擎预热、Bundle 预热、数据预加载
- 其他大厂：暂无相关文章

当然，这里还有一些技术博客上的一些方案也是值得参考的

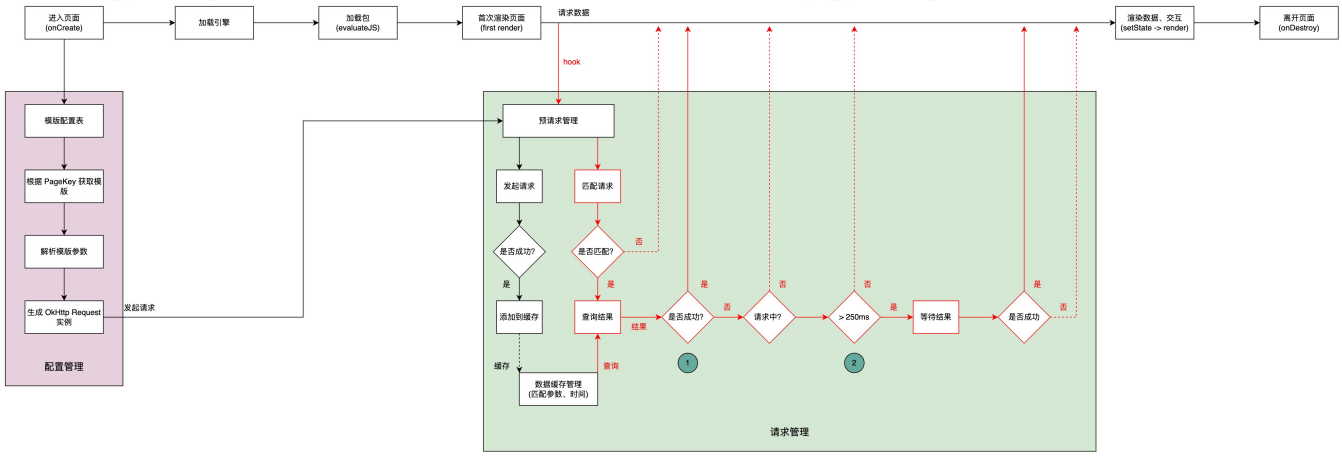


总体思路上还是分两个点

1. 预测性前置请求：当前页预测可能打开的下一个页面，提前请求该页面的数据
2. 跳转时前置请求：跳转与请求并行

我们的方案是和第 2 种思路一致。

3.2 整体设计



第 1 点：对于已经成功的，且结果还在有效期内（目前是 2s）的，直接返回

第 2 点：对于还在请求中，如果已经发出了请求，且请求时间超过了 250ms，就等待结果，如果小于该值，则忽略

缓存有效期 2s 这个没有什么可参考的业内标准，可做成可配置的开关，积累历史经验来定，请求时间 250ms 这个可参考网络请求时间 P50 来定的

3.2.1 配置管理

请求配置主要是配置后端 API 需要的 Http 请求信息，如以下 API 请求中所需要的 method、url 信息

不过，上述接口描述不是完整的 Http 请求信息，一个请求的四大要素

- method：GET / POST
- url：scheme://host:port/path?queries
- headers：k=v
- body：json / byte[]

但这里不需要支持 POST，POST 接口并不是幂等的，所以只支持 GET 即可，body 部分也不需要处理

所以配置表需要满足四大要素的配置，主要有以下信息

类型	字段名	类型	其他说明
页面标识	pageKey	String	页面唯一标识
预请求体	method	String	方法
	baseUrl	String	URL
	path	String	
	queries	Map	
	headers	Map	请求头

配置分为两种

1. **静态配置**
2. **动态模版**

静态配置比较简单，即所有配置均为写死的数据，按这些数据去做网络请求即可，如上述 API 描述中的

- pageno：页码，标识请求第几页，默认第一页写 1
- count：每页的请求个数，默认写 24

但是像 receive_task_id 这个参数是运行时从上一个页面传过来的，没法直接写死，就需要使用动态模版的形式来配置，如

- receive_task_id：{{taskId}}

3.2.1.1 动态模版

动态模版是匹配请求中的动态参数，包括

- url
    - baseUrl
    - query 中的 value 部分
- header 中的 value 部分

动态模版是配置中有动态参数，比如跳转到某个页面，该页面的数据是需要传入一个动态参数去做请求的，这个时候需要用到动态模版。动态模版的设计语法是参考前端的设计：

- Template + Data = Instance：Template 的语法规则是 "{{KEY}}"

如以下模版中有两个需要解析的动态参数 "{{baseUrl}}"、"{{taskId}}"

Template:

模版示例

```
{
    "pageKey":"@shopee-rn/spx-sp-app/SP/InboundDropOff",
    "method":"GET",
    "baseUrl":"{{baseUrl}}",
    "path":"/sp-api/point/dop/receive_task/order/list",
    "queries":{
        "receive_task_id":"{{taskId}}",
        "pageno":"1",
        "count":"24"
    },
    "headers":{
        "accept":"application/json, text/plain, */*"
    }
}
```

Data:

模版数据

```
{
    "baseUrl": "https://sp.spx.test.shopee.tw",
    "taskId": "TW63065694702QT",
}
```

Instance

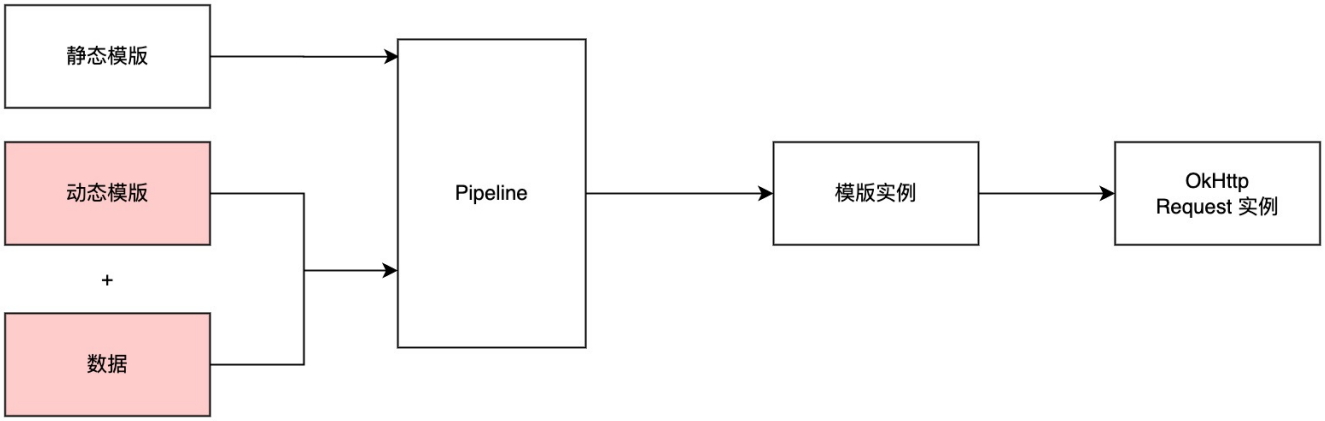
模版实例

```
{
    "pageKey":"@shopee-rn/spx-sp-app/SP/InboundDropOff",
    "method":"GET",
    "baseUrl":"https://sp.spx.test.shopee.tw",
    "path":"/sp-api/point/dop/receive_task/order/list",
    "queries":{
        "receive_task_id":"TW63065694702QT",
        "pageno":"1",
        "count":"24"
    },
    "headers":{
        "accept":"application/json, text/plain, */*"
    }
}
```

那 Data 部分如何获取呢？像 baseUrl 这种参数是整个 App 唯一的，根据国家的不同而静态编码写死的，在运行时设置全局的解析器获取即可，taskId 是页面之间传递的，需要从 Activity 的 Intent 中读取出来解析，所以两者获取方式大致为

- baseUrl：配置全局 KeyResolver，返回 baseUrl 的实际 value
- taskId：配置页面 KeyResolver，返回当前「跳转参数」中响应的字段名

综上，整体流程如下



3.2.2 请求管理

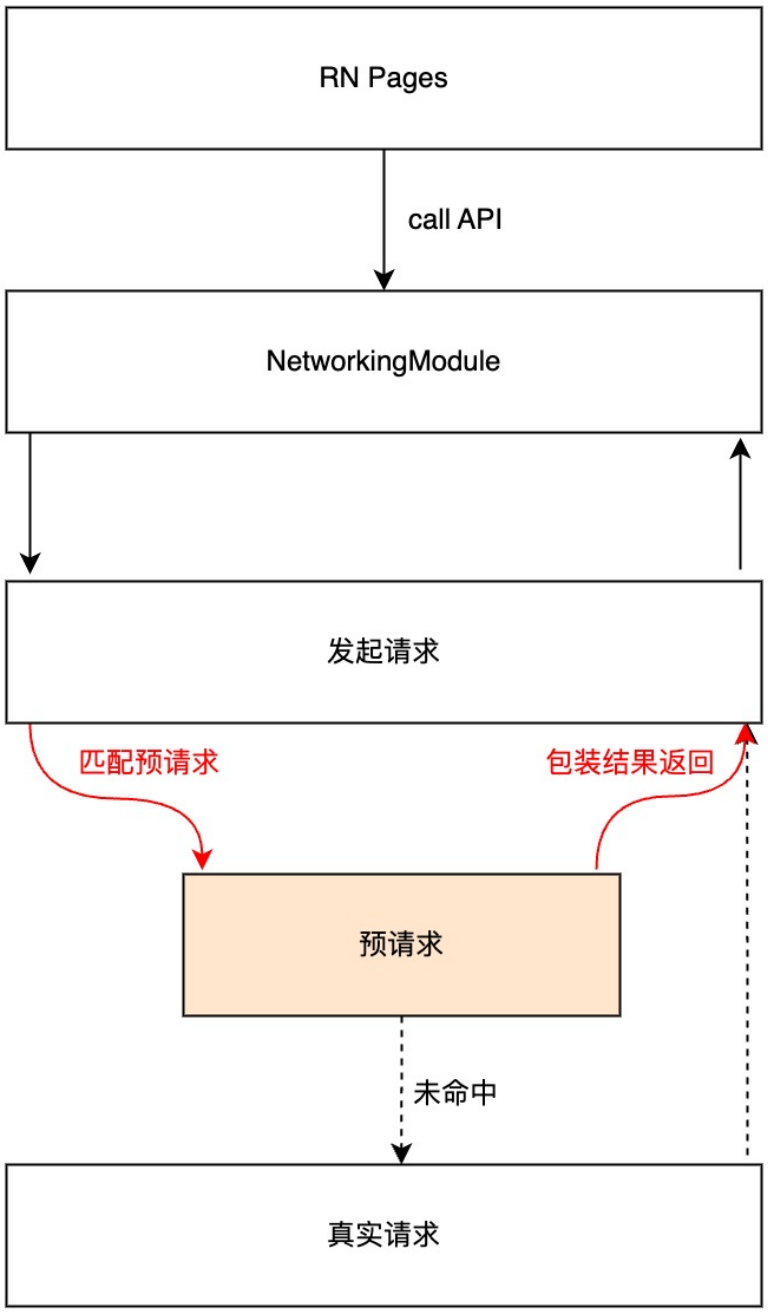
请求管理主要的职能是

1. 发送真实请求获取数据，将数据缓存起来
2. hook 网络请求，查找是否有匹配的结果

这里第一步就是用 OkHttp 去发送请求，不需要再详述，重点 hook 匹配的细节

3.2.2.1 Hook 流程

这里主要是要拦截 RN 的 NetworkingModule 中请求数据的部分



当真实的网络请求发生的时候，我们需要对比真实网络请求和预请求是否一致，这里的一致包括四大部分

- method
- url
  - baseUrl：(scheme://host:port)
  - path
  - queries
- headers

所有 key 和 value 必须相等才可以，这里比较字符串的都是**忽略大小写**的

`method`

- 判断都为 GET / POST

`url 匹配`

- baseUrl：比较字符串

- path：比较字符串
- queries：
    - key 个数相等
    - key 的字符串相等
    - value 的字符串相等

重点是 query 参数，个数必须相等，key 和 value 必须相等，如以下是相等的

- 预请求的 URL：https://sp.spx.test.shopee.tw/sp-api/point/dop/receive_task/order/list?receive_task_id=TW63065694702QT&pageno=1&count=24
- 真实请求 URL：https://sp.spx.test.shopee.tw/sp-api/point/dop/receive_task/order/list?receive_task_id=TW63065694702QT&count=24&pageno=1

headers 匹配

- key 字符串相等
- value 字符串相等
- key 个数可小于真实请求的 key 个数

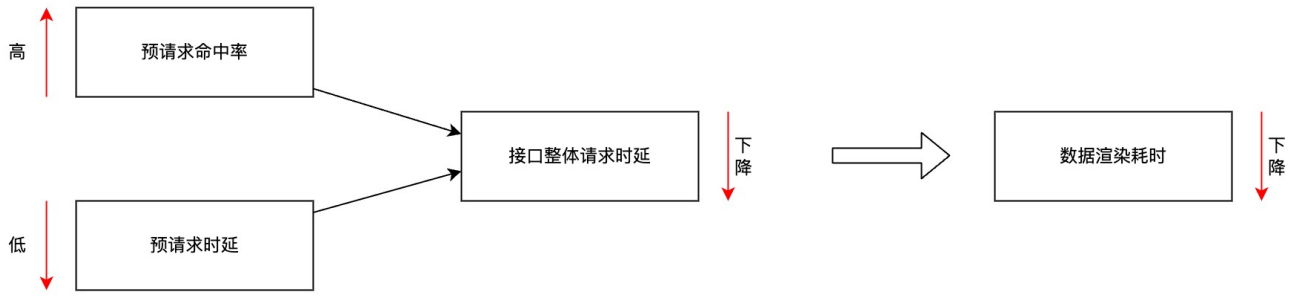
如以下请求可以认为是相等的

- 预请求 Headers
    - "accept":"application/json, text/plain, */*"
- 真实请求 Headers
    - "accept":"application/json, text/plain, */*"
    - "xxx": "xxxx"

3.3 价值衡量与指标设计

如何衡量预请求框架的价值，这里可以从三个方面去看

- 客观指标：框架本身的命中情况
    - 预请求命中率：真实 API 请求匹配到预请求结果的次数 / 预请求总次数
        - 请求发生时已有结果
        - 请求发生时预请求已执行超过 250ms（P50）
    - 预请求时延：预请求返回时间 - hook 时间，这个时间和真实请求的时间比 P 值
        - 请求发生时已有结果的情况，为 0
        - 请求发生时需要继续等待的：预请求返回时间 - hook 时间
    - 接口整体耗时：预请求和真实请求总体聚合在一起看
- 用户层面可感知的指标：最重要的指标
    - 页面加载时间：见 RN 监控



3.3.1 数据协议

类型	字段	含义	说明
公共字段	page_name	页面名称	
	path	URL 中的 path	
事件字段	event_name	事件名	
	value	事件值	
	duration	事件耗时	
	duration_tag	耗时分布	

3.3.2 事件名

- 预请求开始
    - 上报事件：url_request_count
- 预请求命中
    - 上报事件：url_request_matched

序号	指标	事件名	说明
1	预请求命中率	url_request_matched / url_request_count	
2	预请求耗时	url_request_matched 的 duration 字段	

4.推进节奏

时间线	事项	进度	数据准确性验证	其他说明
2022-09-07 ~ 2022-09-20	1. 整理技术方案 2. 按最新方案实现代码逻辑 3. 整理使用文档：使用文档	100%	1. 技术验证 2. 线下自测测试数据	
2022-09-21 ~ 2022-10-09	1. 09.29 日线上发布 2. 补充数据指标及其分析：预请求-数据分析	100% 09-29 live	1. 0929 上午发布，开关打开 2. 1006 晚上关闭开关	
	1. 细化预请求耗时分布 2. 预请求的接口成功率统计方式优化：不再影响接口网络成功率和业务成功率 3. 重新使用搜索页面来测评预请求框架的效果	10.14 live		
2022-10-10 ~	1. 统计搜索页接口：搜索页预请求分析			

5.结果

5.1 指标查询与报表分析

数据查询平台：https://mdap.shopee.io/boussole/ssc-spx-servicepoint-dashboard/dashboard/view/wO_2rQI1NHASX7Lc?mode=preview

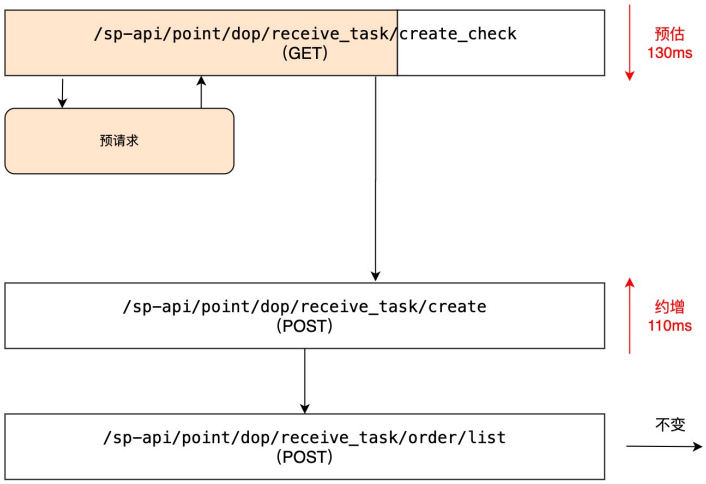
5.1.1 Case 分析一

API 配置：/sp-api/point/dop/receive_task/create_check


```
{
  "pageKey": "@shopee-rn/spx-sp-app/SP/InboundDropOff",
  "method": "GET",
  "baseUrl": "{{baseUrl}}",
  "path": "/sp-api/point/dop/receive_task/create_check",
  "headers": {
    "accept": "application/json, text/plain, */*"
  },
  "waitTime": 200
},
```

- 预请求命中率：64.6% (09.29~10.05)
- 预请求耗时：94.2ms（https://docs.google.com/spreadsheets/d/13W2XXAFAu10yuYglXvCEYbgVji8jxmi1bNslXBIKUws/edit#gid=353246747）
- 接口整体耗时：没法单独统计
- 数据渲染时间：降低 50ms（低于预期）（https://docs.google.com/spreadsheets/d/1t_0DP2CGQuxuksgqSbBWbUwI0WuYI3PVCWx3zyKUWN4/edit#gid=446752335）
    - 秒开率：提升 0.7%（低于预期）

原因分析：OkHttpClient 的连接池复用机制导致



5.1.2 Case 分析二

API 配置：/sp-api/point/collection/parcel_search/list


```
{
  "pageKey": "@shopee-rn/spx-sp-app/SP/ParcelSearch",
  "method": "GET",
  "baseUrl": "{{baseUrl}}",
  "path": "/sp-api/point/collection/parcel_search/list",
  "queries": {},
  "headers": {
    "accept": "application/json, text/plain, */*"
  },
  "waitTime": 100
}
```

- 预请求命中率：10.6% (10.19)
- 预请求耗时：95ms (https://docs.google.com/spreadsheets/d/13W2XXAFAu10yuYglXvCEYbgVji8jxmi1bNslXBIKUws/edit#gid=1283319592)
- 接口整体耗时：没法单独统计
- 数据渲染时间：降低 50ms (https://docs.google.com/spreadsheets/d/1ghkygmQ5VgdTD_HY4i1mKyfXOt6yZAoKRaFDwX4SolY/edit#gid=476614914)
    - 秒开率：提升 4%

5.2 结论

- 从命中率方面看，inbound-drop-off 页面命中率为 64.6% 之间，也就意味着从进入页面到第一个请求发出，已经过了 200ms 左右，基本上可以论证其价值
- 从预请求耗时方面看 (发生真实请求的时间点为起点，预请求结果返回为终点)
    - avg：从 290ms 下降到 94.2ms
    - P95：从 657ms 下降到 300ms 以内
- 从页面加载时间看，parcel_search 页提升 4%，达到 90% 秒开率的标准

6.业内参考

移动端页面秒开优化总结

前端性能优化的总结