Rexxar分享的内容

- 1.什么是混合开发
- 2.Rexxar是什么
- 3.为什么要引入Rexxar
- 4.Rexxar的功能与特点
- 5.Rexxar的组成
- 6.Rexxar在移动端如何加载、解析、渲染web模板
- 7.Rexxar的使用流程
- 8.Rexxar使用过程中的限制及注意事项
- 9.Rexxar与其他混合开发框架的对比
- 10.总结与展望

混合开发（Rexxar）

混合开发是 Native 和 Web 技术一起用，开发者以 Native 代码为主体，在合适的地方部分使用 Web 技术。比如在 Android 中的 Activity 内放置一个 Webview（一个浏览器引擎，只拥有渲染 HTML，CSS 和执行 JavaScript 的核心功能）。这样，部分用户界面就可以在 WebView 中使用 Web 技术实现。

促使我们在移动开发中使用 Web 技术主要动力在于，相比于 Native 技术，Web 技术具有诸多优势：

- 高效率的界面开发：HTML，CSS，JavaScript 的组合被证明在用户界面开发方面具有很高的效率。
- 跨平台：统一的浏览器内核标准，使得 Web 技术具有跨平台特性。iOS 和 Android 可以使用一套代码。
- 热更新：可越过发布渠道自主更新应用。

这些优势都和开发效率有关。Web 技术具有这些优势的原因是，Web 技术是一个开放标准。基于开放的标准已经发展出来庞大生态，而且这个生态从 PC 时代发展至今已积累多年，开发者可以利用生态中产出的各种成果，从而省去很多重复工作。

在大型移动应用的开发中，项目代码庞杂，通常还需要 iOS，Android，移动 Web 和 桌面 Web 全平台支持。相对于同时开发几个版本，使用混合开发显然可以在代码重用、开发成本和效率方面有很大的优势，在权衡性能体验的前提下，使用混合开发是非常现实的选择。

Rexxar 是什么

Rexxar是一个针对移动端支持Android的混合开发框架，它是基于豆瓣Rexxar-Android框架修改的。是移动端Web资源的离线缓存解决方案，能够拦截webview的请求，并优先使用本地缓存静态资源进行响

应，以此来对webview加载页面性能进行优化。

为什么用Rexxar

- 解决webview的资源缓存问题
- 解决离线资源缓存与更新的通用问题

Rexxar的功能与特点

1.解决webview的资源缓存问题

- 本地预缓存Web资源
- 通过重写shouldInterceptRequest方法进行请求透明拦截，优先使用本地资源进行响应，提升页面响应速度，节省流量
- 资源缓存策略的配置与管理，对缓存资源具有更强的控制力
- 离线资源打包到安装包，可以做到首次请求不走网络

2.解决离线资源缓存与更新的通用问题

- 资源的版本管理
- 资源的下载与更新
- 项目构建时web资源自动打包

Rexxar的组成

Rexxar主要由三部分组成：

- ResourceProxy：资源代理。负责资源管理，比如获取缓存的资源，写入缓存资源，请求线上资源。
- Route：管理路由表，负责刷新路由，存储缓存的路由，加载本地的路由。然后通过uri找到对应的html页面，一条Route包含一个uri的正则匹配规则和一个html地址。
- HybridWebView：增强版WebView，封装了一些WebView的Cookie的同步，WebView加载状态监听回调的处理，页面load完成后数据的获取和所需要传递的参数等。

ResourceProxy

ResourceProxy负责资源管理，空闲时间下载html文件然后缓存，读取预置到asset文件和缓存中的资源，请求线上资源等。

Route

Route 路由表，我们使用了一个 json 文件来表达路由表，根据route地址，请求route然后返回route的原始内容，通过缓存Route列表，配置Route策略等操作来管理Route。


```
{
  "items": [
    {
      "deploy_time": "Thu, 09 Nov 2017 03:11:15 GMT",
      "remote_file": "https://raw.githubusercontent.com/pxfile/Hybird/master/polygon.html",
      "uri": "http://backend.igengmei.com/hybrid/topic_reply/19227490[/]?.*"
    },
    {
      "deploy_time": "Thu, 10 Nov 2017 03:11:15 GMT",
      "remote_file": "https://raw.githubusercontent.com/pxfile/Hybird/master/polygon.html",
      "uri": "http://backend.igengmei.com/hybrid/topic_reply/19227490[/]?.*"
    }
  ],
  "deploy_time": "Thu, 09 Nov 2017 03:11:15 GMT"
}
```

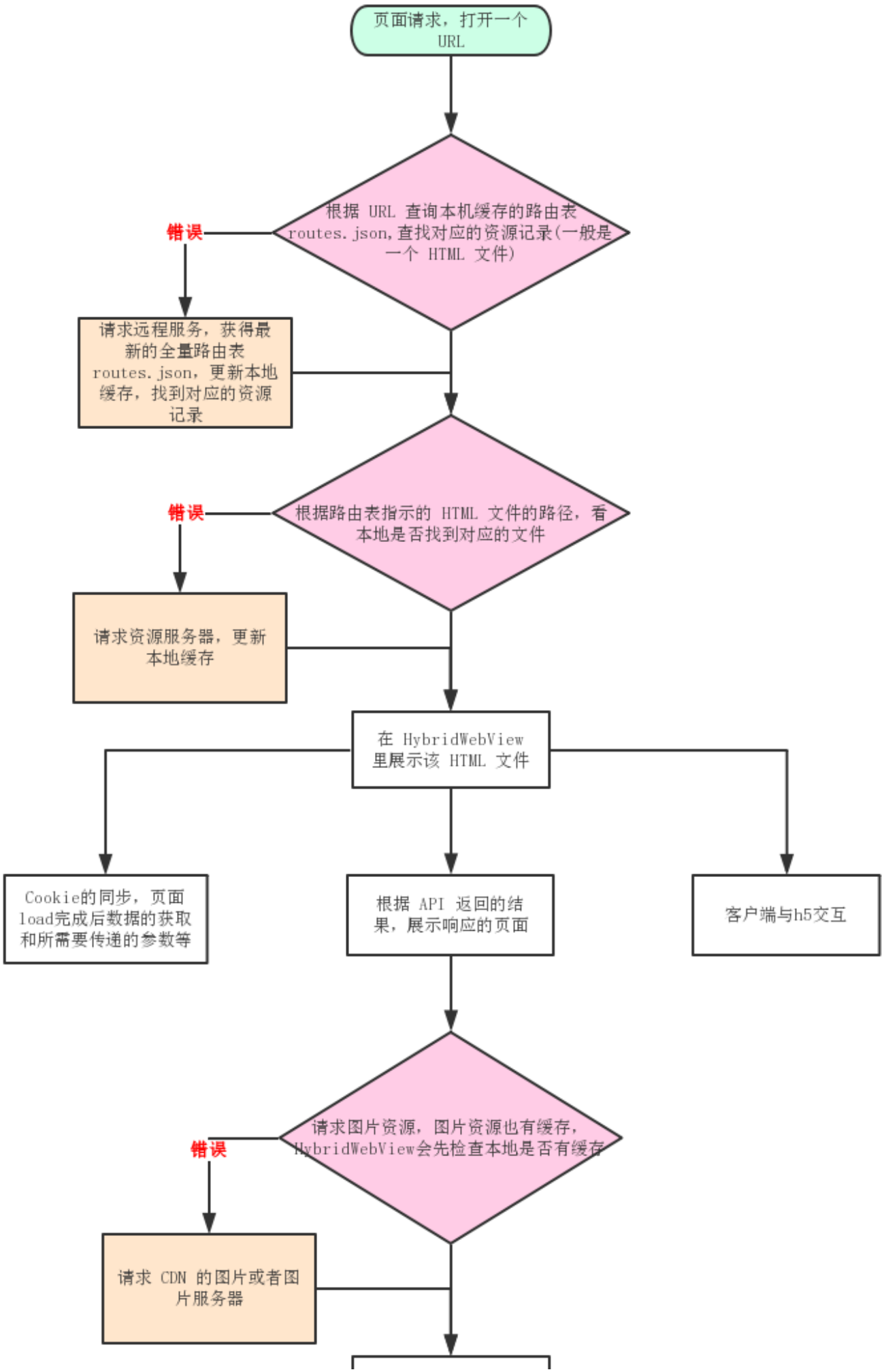

发布的每个版本的 App 安装包都会包含最新版本的 routes.json 文件。在 App 启动时，都会尝试下载最新版本的 routes.json。在遇到无法解析的 URL 时，也会去下载新版 routes.json,或者根据 URL 查询本机缓存的路由表 routes.json不能找到对应的资源记录时，也会请求远程服务器获得最新的全量路由表 routes.json，更新本地缓存。

HybridWebView

HybridWebView 是提供了一个运行前端代码的容器。它也是一个内嵌的浏览器（WebView）。不是只简单的load一个 URL 地址，还对内嵌的浏览器做了很多开发，为其包装了很多附加功能。

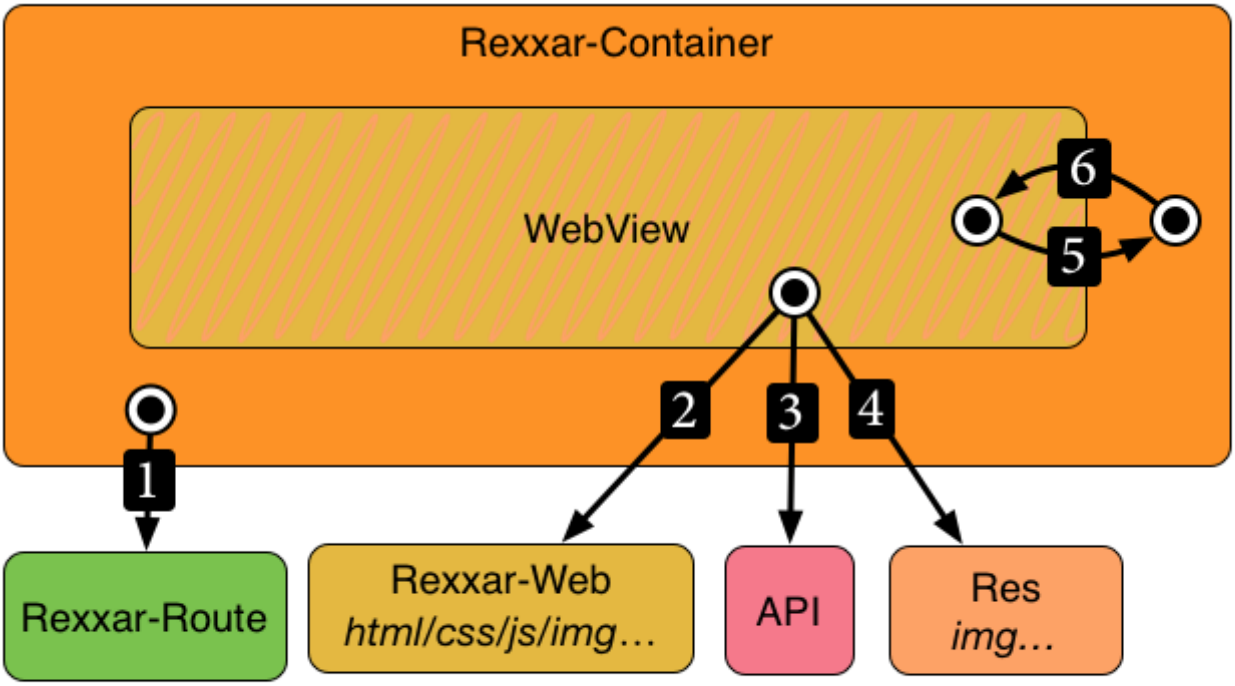
Rexxar在移动端如何加载、解析、渲染web模板

Rexxar 的工作流程图


页面请求，打开一个URL
根据 URL 查询本机缓存的路由表routes.json,查找对应的资源记录(一般是一个 HTML 文件)
错误
请求远程服务，获得最新的全量路由表routes.json，更新本地缓存，找到对应的资源记录
根据路由表指示的 HTML 文件的路径，看本地是否找到对应的文件
错误
请求资源服务器，更新本地缓存
在 HybridWebView 里展示该 HTML 文件
Cookie的同步，页面load完成后数据的获取和所需要传递的参数等
根据 API 返回的结果，展示响应的页面
客户端与h5交互
请求图片资源，图片资源也有缓存，HybridWebView会先检查本地是否有缓存
错误
请求 CDN 的图片或者图片服务器

展示页面图片

Rexxar 页面执行过程



Rexxar 的工作流

客户端接到一个页面请求，要打开一个 URL：
http://backend.igengmei.com/hybrid/topic_reply/19227490。

- 1.根据 URL 查询本机缓存的路由表 routes.json，看是否能够找到对应的资源记录(一般是一个 HTML 文件)。如果找到不到，请求服务，获得最新的全量路由表 routes.json，更新本地缓存，找到对应的资源记录；
- 2.根据路由表指示的 HTML 文件的路径，看本地是否找到对应的文件。如果找不到，请求资源服务器，更新本地缓存；在HybridWebView里展示该 HTML 文件；如有需要，会在 HybridWebView 中请求图片资源，图片资源也有缓存，HybridWebView会先检查本地缓存。如不存在，会请求 CDN 的图片或者图片服务器；
- 3.前端代码在HybridWebView里继续执行，发出 API 请求。HybridWebView代理这些请求，WebView的Cookie的同步，WebView加载状态监听回调的处理，页面load完成后数据的获取和所需要传递的参数等；
- 4.前端代码继续执行，根据 API 返回的结果，展示响应的页面，可能会请求 CDN 的图片或者图片服务器等；

- 5.前端代码继续执行，js和native的交互；

Rexxar的使用流程

配置

1. 初始化

在Application的中调用

```
    /**
     * 配置HybridWebView
     */
    private void initHybridWebView(Context context) {
        Rexxar.setConfig(new Rexxar.Config().
                        setContext(context).
                        setAsyncLoadRoute(true).
                        setOkHttpClient(new OkHttpClient().newBuilder()
                                .retryOnConnectionFailure(true)
//                              .addNetworkInterceptor(new AuthInterceptor())
                                .build()).
                        setRouteConfig(new RouteManager.RouteConfig( routeApi: "https://raw.githubusercontent.com/pxfile/Hybird/master/routes/routes.json")).
                        setUserAgent(com.wanmeizhensuo.zhensuo.common.webview.core.Constants.USER_AGENT).
                        setWindowClient(com.wanmeizhensuo.zhensuo.common.webview.core.Constants.WINDOW_CLIENT).
                        setProtocolName(com.wanmeizhensuo.zhensuo.common.webview.core.Constants.PROTOCOL_NAME)
        );
    }
```

2. 设置路由表文件 api：

```
RouteManager.getInstance().setRouteApi("https://raw.githubusercontent.com/pxfile/Hybird/master/routes/routes.json");
```

Rexxar 使用 uri 来标识页面，提供一个正确的 uri 就可以打开对应的页面，路由表提供了每个 uri 对应的 html 资源的下载地址。

Demo 中的路由表如下：

```
{
  "items": [
    {
      "deploy_time": "Thu, 09 Nov 2017 03:11:15 GMT",
      "remote_file": "https://raw.githubusercontent.com/pxfile/Hybird/master/polygon.html",
      "uri": "http://backend.igengmei.com/hybrid/topic_reply/19227490[/]?.*"
    },
    {
      "deploy_time": "Thu, 10 Nov 2017 03:11:15 GMT",
      "remote_file": "https://raw.githubusercontent.com/pxfile/Hybird/master/polygon.html",
      "uri": "http://backend.igengmei.com/hybrid/topic_reply/19227490[/]?.*"
    }
  ],
  "deploy_time": "Thu, 09 Nov 2017 03:11:15 GMT"
}
```

3. 设置需要代理或缓存的请求host

```
private static final List<String> PROXY_HOSTS = new ArrayList<>();

static {
    PROXY_HOSTS.add("raw.githubusercontent.com");
}
```

```
// 设置需要代理的资源
ResourceProxy.getInstance().addProxyHosts(PROXY_HOSTS);
```

Rexxar是通过`WebViewClient`的`shouldInterceptRequest`方法来拦拦截请求，请求线上数据并返回给'webview'。为了减少不必要的流程破坏，只有明确需要拦截的hosts（支持正则）的请求才会被拦截代理，并根据mime-type决定哪些内容需要缓存。

4. 预置资源文件

使用 Rexxar 一般会预置一份路由表，以及资源文件在应用包中。这样就可以减少用户的下载，加快第一次打开页面的速度。在没有网络的情况下，如果没有数据请求的话，页面也可访问。这都有利于用户体验。
预置文件路径是`assets/rexxar`，暂不支持修改。

使用 HybridWebView

直接使用`HybridWebView`为的混合开发客户端容器。或者也可以在`HybridWebView`基础上实现你自己的客户端容器。

为了初始化 HybridWebView，你需要只一个 url。在路由表文件 api 提供的路由表中可以找到这个 url。这个 url 标识了该页面所需使用的资源文件的位置。HybridWebView会通过 url 在路由表中寻找对应的 javascript，css，html 资源文件。

```
mHybridWebView.loadUri("http://backend.igengmei.com/hybrid/topic_reply/19227490");
```

Rexxar使用过程中的限制及注意事项

- 1.性能方面

Web 的性能没法和 Native 相比。这种状况可能会长期存在。因为，前端代码运行于内嵌浏览器之上，和直接调用原生系统相比，理论上总会存在性能上的差距，可以用规避的方式面对性能问题：即性能问题会明显影响到用户体验时，我们就不使用 Rexxar 来做，而是使用传统 Native 写两份代码，一份 iOS，一份 Android。这样就限缩了 Rexxar 的使用范围。

- 2.内存问题

Rexxar在客户端的实现其实就是一个定制了更多功能的WebView。由于Rexxar使用的是系统的WebView。所以对App的体积没有影响。但是Rexxar同时使用很多个WebView带来的内存问题，这是需要注意的。

- 3.错误报告

Rexxar的Crash有两种：

- 一种是JavaScript的错误，也就是应用逻辑的问题。这类错误在WebView中做了捕获，然后通过App的日志系统发回服务器。
- 一种是WebView的Crash，这种错误WebView自己无法捕获，现在是通过bugly这种原生的Crash收集系统收集。

豆瓣在应用中使用 Rexxar 之后，在收集到的 Crash Report 中，JavaScript 的相关错误，和浏览器相关的错误开始增加。而对这类错误，由于移动应用的使用环境更为复杂，错误报告经过了 JavaScript 引擎，原生系统两层之后，给出的错误信息并不够明确。豆瓣在这方面的经验也并不多，现在还没有很好的办法降低这类错误。这对提高 App 的稳定性带来了问题。

为什么不用PhonGap/Cordova

在混合开发中早已有了很成熟的方案，就是PhoneGap和它的后继者Cordoba. 为什么豆瓣还要造自己的轮子呢?

如果Rexxar方案定义为前端和原生技术的混合使用，那他们认为PhoneGap/Cordova严格来说不算是Rexxar方案，因为它的目标是全面使用前端技术开发移动应用，而不是前端和原生技术混合使用。但是，包括Cordova，还可以加上React Native，以及Rexxar的目标是一致的：使用前端技术来开发移动应用，提高工程效率。

豆瓣实际上使用PhoneGap开发过一款移动App，并在AppStore上架了，这个应用叫豆瓣音乐人，因此，其实豆瓣对PhoneGap/Cordova已经有一定了解和使用经验。为何在开发豆瓣App时又造了一个叫Rexxar的“轮子”呢？这是因为，他们对PhoneGap/Cordova这个项目的理念并不完全赞同，Rexxar的出发点和PhoneGap/Cordova并不一样。

PhoneGap/Cordova这个项目极具野心。它希望完全使用前端技术完成移动开发。所以，可以看到它尽力让前端技术完成尽量多的开发工作，只在前端无法直接调用的原生系统功能方面提供了前端可用的接口。主流的PhoneGap/Cordova项目将业务逻辑都实现在一个WebView中。目标是，让开发者只使用前端技术就可以完成一个移动应用的所有开发工作。这种做法需要有一个前提：前端技术可以解决移动开发的所有需求。他们认为PhoneGap/Cordova这个理念在现阶段有些过于理想化了，或者说过于激进了。

Rexxar则相对实际，或者说保守一些。他们仍然认为，**现阶段，甚至在相当遥远的未来，移动开发中前端技术都不太可能完全代替原生技术**。但他们同时承认，**移动开发中总是存在部分功能是适合使用前端技术完成的**。在他们的认识中，前端技术和原生技术应该是共存的。移动开发中，前端技术不会完全代替原生技术；而有了前端技术的加入，移动开发的效率会提高。基于这种认识，豆瓣开发了Rexxar。

可以看到，**Rexxar立足于在一个原生项目使用前端技术，而不是整个项目都使用前端技术实现**。他们甚至提供一个页面部分使用Rexxar完成，部分使用原生技术实现的方案。豆瓣希望借助前端技术优秀的排版能力、开发速度、通用性，来弥补原生开发在这方面的不足。在微信作为主要内容分享渠道的今天，这样做还带来了一个额外的好处，Rexxar页面可以平滑的使用在微信中。

总结而言，如果Rexxar和PhoneGap/Cordova比较的话，大目标是一致的：使用前端技术开发移动应用。实现技术栈差不多：使用WebView，提供调用原生功能的接口。但是，出发点不一样。PhoneGap/Cordova致力于完全使用前端技术进行移动开发；Rexxar致力于在移动项目中部分使用前端技术。

鼓励移动开发者学习前端技术

目前，我们移动团队大约有十位客户端工程师，其中 iOS 和 Android 各一半。可以委派一位优秀的前端工程师专门支持App中的混合开发，他负责Rexxar Web的开发，提供基础设施。同时如果有一些较复杂的业务要用Rexxar实现，他也会参与和指导业务开发。

使用Rexxar这类混合开发技术，使得团队开发的技术栈向前端技术偏斜了。所以，较理想的配置是团队中加入较优秀的前端工程师，由他来处理基础设施的开发，和疑难问题的解决。同时，整个团队需要理解混合开发所带来的优势，认可这个开发方式的转变，并且愿意学习和调整自己的技术栈。

在项目中，在合适的场景中，可以优先使用Rexxar。在团队中，应该鼓励非前端工程师学习和使用前端技术。由于以前专门组织了关于前端技术内部培训，让有意愿的非前端工程师具有了可以使用前端技术进行日常开发的基本能力。期望在App的日常开发中，大部分Rexxar页面都可以由客户端工程师完成，前端工程师会帮忙做Code Review和解决疑难问题。

总结与展望

通过在移动开发中使用Rexxar，在一定程度上提高了开发效率。以前一个页面需要 iOS 和 Android 两位工程师各开发一遍，现在只需要一位工程师写一次前端代码，甚至还可以应用到移动 Web 站上去。前端技术在开发界面方面也有效率上的优势，热部署能力，使他们规避了发布移动应用的审核过程，也让bug修复过程更便利。

豆瓣将Rexxar这个项目开源，一方面，是因为提高移动开发的工程效率是一个普遍问题，而他们的实践结果也证明Rexxar确实帮助改善了工程效率。所以，他们认为Rexxar应该能给大家提供一些借鉴的方向。另一方面，是为了提高项目本身的质量，没有方案是完美的，Rexxar也还存在不少问题。开源这个项目，促使他们提高了整个项目的代码质量。同时，也更容易听到大家的意见和建议。

虽然Rexxar仍然存在一些问题和使用上的限制。但是在有限的使用中，豆瓣App团队仍然收获不少。在未来他们会持续推动Rexxar在豆瓣移动开发中的使用。对于Rexxar未来的发展，他们主要关注两个方面：

- 一方面是基础设施，比如，如何在产品中，更好地监控Rexxar页面出现的问题，如何调试和解决Rexxar页面出现的bug。如果希望在大型项目中使用Rexxar，这些基础设施是应该配备的；

- 另一方面是性能，Rexxar仍然跑在浏览器引擎中。浏览器引擎这个中间层提高了工程效率，但也因为性能问题局限了其使用范围。所以，他们会花一些精力提高Rexxar的运行效率。比如，Rexxar的iOS版一直在关注从UIWebView迁移到WKWebView的可能性。