计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):19−27 [doi: 10.15888/j.cnki.csa.006038]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Web 技术进行移动应用开发和中间件的研究①

罗宏俊, 冯　瑞

(复旦大学 计算机科学技术学院, 上海 201203)
(复旦大学 上海市智能信息处理重点实验室, 上海 201203)

摘　要: 提出了一种基于纯 Web 技术来进行移动应用开发的新型开发模式, 并指出其可行性和意义. 基于这样的一种开发模式, 通过 MUI 和 HTML5+的技术实现一个跨平台的移动端体育竞技平台. 最后打包部署在包括 ios、Android 手机与平板等所有移动端设备上运行, 验证了这样的一种开发模式的可行性与优势所在. 在 Web 技术日益发达的今天, 这样的开发方式会得到越来越广泛的关注和应用.

关键词: Web 技术; Web 应用; 移动应用; MUI; HTML5+

引用格式: 罗宏俊,冯瑞.基于 Web 技术进行移动应用开发和中间件的研究.计算机系统应用,2017,26(11):19–27. http://www.c-s-a.org.cn/1003-3254/6038.html

Research on Mobile Application Development and Middle Controller Based on Web Technology

LUO Hong-Jun, FENG Rui

(School of Computer Science, Fudan University, Shanghai 201203, China)
(Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai 201203, China)

Abstract: This paper comes up with a mode of development to build a mobile application with Web technology, together with its meaning. Based on this kind of mode, using MUI and HTML5+, we could build a cross-platform mobile sports competition platform. At last, to verify the feasibility and advantage of this mode of development, we package and deploy the application to run on mobile devices such as ios, Android System and pad. Since Web technology has become highly developed, this mode of development will be the focus and used more and more widely.

Key words: Web technology; Web application; mobile application; MUI; HTML5+

引言

在 Web 技术日益发达的今天, 前端技术能够实现的事情越来越多, 包括使用以 Node-Webkit 或者有道 Hex 为代表进行桌面应用的开发, 使用 NodeJS 来进行后端程序开发, 通过游戏引擎和对应的 IDE 来使用 JS 进行游戏开发, 使用 WebGL 能够进行图形图像处理等等. 对比后台服务器开发, 前端也逐渐建立起完善的自动化开发工作流, 例如 Grunt、Gulp 等, 和模块化开发方案 Webpack 和 browserify. 这一系列的技术方案, 映射出 Web 前端开发的高速和蓬勃发展. 由于 Web 技术具备良好的跨平台性, 可以在 ios、Android、windows、mac、linux 等多个环境都能运行良好, 开发成本相较于传统的平台原生开发要低, 在需要迅速迭代开发运行项目都有着先天的优势, 如果能有一种中间件技术, 使开发者能够以 Web 技术进行开发并打包生成不同平台的应用, 并流畅运行, 一定能够迅速占领市场, 为大多数开发者和企业所采用. 本文要讨论的技术方案, 是通过中间件技术的支持, 使用 Web 技术来进行移动应用的开发和优化研究[1], 并打包分发到各个平台流畅运行.

① 基金项目: 国家科技支撑计划 (2013BAH09F01); 上海市科委科技创新行动计划 (14511106900); 临港地区智能制造产业专项 (ZN2016020103)
收稿时间: 2017-02-16; 修改时间: 2017-03-02; 采用时间: 2017-03-09

首先需要理解 Web App, Native App, Hybrid App[2]. 随着移动互联网的不断发展, 越来越多的开发人员都偏向于移动应用的开发, 开发者应该选用何种技术方案进行移动应用的开发成为越来越多企业需要考虑的问题.

1) Web App. 以浏览器为载体, 通过在浏览器访问网址, 以下载整个 Web App 来进行交互, Web App 的外观和体验都在往 Native App 靠拢, 其不需要通过市场审核快速上线的优点使很多企业采取这样的开发方式. 自 HTML5 的兴起以来, 表现能力与调用设备原生性能的能力也大大提升, 缓存功能的加入使得 Web App 的出现成为一种可能. 不足之处在于, Web App 始终需要在浏览器运行, 依赖网速和 App 体验感等都成为其最难突破的问题.

2) Native App. 对于大企业而言更关注的是性能与体验, 在不考虑成本预算和分发渠道的前提下, 都会选择进行原生应用开发, 也即 Native App, 此类 App 需要进行 IOS 开发, Android 开发, 为了更多渠道的分发和吸引用户的需要, 还需再进行 WAP 开发和流应用开发, 才能最大程度上进行全渠道用户的覆盖和吸引. 而每一次新功能的更新都需要在 Appstore 和 Android 各类应用市场通过上传新的 App package 以使用户获得更新通知, 每一次的更新都需要进行数天的时间来进行审核, 降低了新功能上线的及时性. 尽管如此, 通过市场渠道的分发, 使得 Native App 的管理更加规范.

3) Hybrid App. 混合型应用, 通过 Native 与 Web 技术的协作, 生成一套 JS SDK 给予了前端操作访问 Native 资源的权限, 前端通过将静态资源部署到 Native 框架中, 最后打包生成原生 App, 实现了一套代码, 多端使用的目的[3]. 其优势明显, 利用有限的开发资源, 来快速开发出多端的应用, 用以抢占市场, 是非常适合新业务试错的开发模式. 其缺点聚焦于性能问题, 还是需要不断的优化才能使应用的表现趋近于 Native App. 然而, 随着 Web 技术的不断发展, Hybrid App 与 Native App 的差距越来越少, 也吸引了越来越多的开发者和公司. 其中比较有代表性的 Hybrid 开发框架有 DCloud 出品的 MUI&HTML5+、开源的 Phonegap &Cordova、阿里出品的 MSUI、腾讯出品的 FrozenUI 以及正益无线出品的 AppCan 等产品[4].

而随着 Android 和 iPhone 手机的流行, 如果开发人员想要开发出多个平台的移动应用, 就必须要精通多种语言, 而由于多个平台的不兼容性, 没有统一的接口和语言来进行移动应用开发的话, 开发成本会大大的提高, 所以选择 Hybrid App 成为了越来越多公司的主流的开发方案. 而围绕着这几个概念, 本文将聚焦于 Hybrid App 的开发, 叙述 Hybrid App 实现的原理, 以及逼近 Native App 性能的优化工作.

1 基于中间件技术实现 Hybrid App

在 Web 技术日益发达的今天, 业务的迅速发展对开发效率和成本控制提出了更高的要求, 此时使用 IOS&Android 进行原生应用的开发显得成本过高, 而 HTML5 的低成本, 高效率和跨平台等特性使得使用中间件技术开发的 Hybrid App 的开发方式迅速崛起和流行, 而实现 Hybrid App 的开发, 上层使用 Web 技术 HTML、CSS 和 JS 进行表现层开发, 需要 Native 层提供接口访问, 以将 Native 硬件访问的权限开放给上层.

实现 Hybrid App 这种开发模式, 需要考虑以下几点:

1) App 的 Native 层和 Web 层各自的工作是什么?

2) Native 层与 Web 层交互接口如何设计?

3) 资源的存储与避免白屏等技术工作.

1.1 Native 层和 Web 层的功能

App 的 Native 层和 Web 层功能不一致, Native 层负责提供一个宿主环境, 以及本地功能性接口予 Web 层访问, 且必须考虑好与 Web 层的交互, 否则会影响后续的维护. 而 Web 层则根据需求实现相应的业务功能, 合理调用 Native 层提供的接口. 比如: Web 层对定位功能、重力感应、通信录、文件管理等 Native 功能调用, Native 与 Web 页面的相互跳转等. 如图 1 所示.

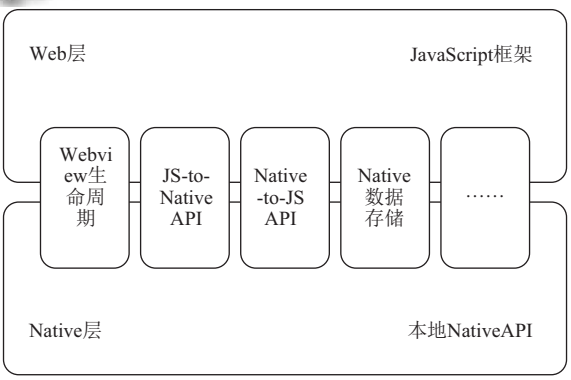

图 1 Native 层与 Web 层功能交互

1.2 Native 与 Web 层交互接口的设计

Native 层与 Web 层的交互是通过 Native 层调用 Web 层的 JS 方法和 Web 层通过 Native 层封装的 JS

API 接口方法来访问 Native 资源[6]. 事实上, App Native 层可以注册 URL 在调度中心以供 App 调用, 而封装的 API 接口方法则是通过创建此类 URL 以供 Native 层捕获以进行通信, 如: QQ 的 url 是 (mqq://), 微信是 (weixin://), 淘宝是 (taobao://).

两者的交互通过将 API 方法绑定在 Web 层的 window 对象上面, App 通过原生的方法调用绑定在 Webview 的 window 对象的 JS 方法以实现对 Web 层的通信; 而关于 Web 层对 Native 层的通信则通过 App 实现对 Web 层的 watcher 模式, 每当 Webview 发生了改变, 该 Webview 所发出的 URL 的哈希值就会发生改变, 此时 App 则会通过监控哈希值的改变来进行对应的操作.

这里 API 的设计与 ajax 的机制非常类似, 开发者只需要通过这种方式来设计和创建 ajax 请求即可. 这里约定的 Native API 设计格式为:

```
RequestHybrid({
    tagname: 'hybridAPI',
    param: {},
    callback: function(data) {}
});
```

执行之后即形成一个这样的 url:

hybridschema://hybridAPI?callback=hybridcallback&param=*******.

在这样的机制下开发者就可以设计各类 Web 层需要访问 Native 层的 API 了. 如图 2 所示.

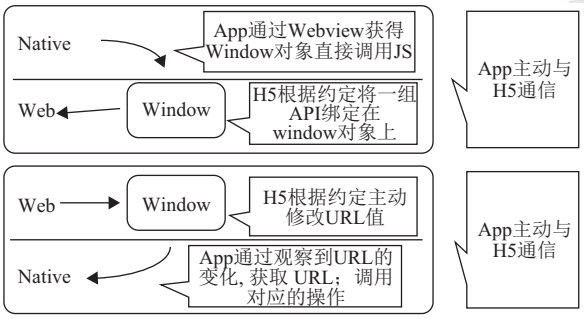

图 2 Native 层与 Web 层交互原理

1.3 资源的存储与白屏的避免

资源的存储分为文本的存储与文件的存储, 使用 HTML5 的 localStorage 与 sessionStorage 甚至 indexedDB 均可, 但是不可在应用内跨域操作, 文件的大小一般也有限制, 因此这里可以封装本地的存储 API 以供 Web 层使用, 去除了跨域和文件大小的限制, 但是却引来了轻微的延时.

白屏问题是 HTML5 带来的性能问题. HTML5 的性能相比于原生的低下, 例如切换页面时白屏, 刷新, 加载有延迟等. 由于浏览器页面切换的机制是跳转到下一个页面后, 需要先联网, 载入页面代码, 构建 DOM 树和 render 树, 最后进行显示, 在结果渲染完毕之前, 会出现短暂时间的白屏.

白屏分为创建 Webview 页面过慢引发的白屏和数据加载渲染页面过慢产生的白屏, 解决方案是封装原生的 Webview 对象, 然后通过 JS 来控制 Webview 窗体的切换, 从而实现原生操作 Webview 的方式. 另外 Web 层的解决方案有现载、预加载与预截图几种[7].

现载方案需要一定的时间来渲染 Webview, 此时当超过一定时间还没渲染完毕则通过调用上面封装的本地 loadingUI 控件以给予用户响应时间, 掩盖白屏卡顿, 通过交互设计的方式来给予用户更好的体验.

预加载方案通过在用户空闲时预先加载好页面, 使其预先渲染完成, 当用户点击之后, 将预先加载的页面显示出来, 免去了渲染的过程, 直接避免了白屏的发生.

预截图是通过在预加载的过程中将窗体截图, 存储于 Bitmap 对象, 这时当切换窗体的时候, 只需要将预先截图的 Bitmap 对象进行动画切换, 则可掩盖联网获取数据时画面重绘的卡顿感了.

2 移动端“GMF 运动”的设计与实现

现需要进行“GMF 运动”移动平台的设计, 需要根据产品需求设计开发出 iPad 平板端, iPhone 各型号 ios 端, Android 各型号 Android 端, 需要添加定位功能, 支付功能, 缓存等功能. 由于产品的完备性, 使得 App 对 Native 功能的需求度上升, 并且也对性能方面也有一定的需求. 通过原生 Native 方式进行开发需要通过 java 和 c 的方式并对相应性能进行优化, 会在很大程度上增加产品的开发周期和学习成本.

根据前文的分析, 开发者可以利用 Hybrid App 的中间件技术来解决这一问题, 通过 Web 技术来开发同一套代码, 不仅能很好的解决各端的适配问题, 也能缩短产品的开发周期和学习成本, 通过对 Web 的优化, 在性能体验上也会比较优秀.

根据国内外 Web 移动应用开发中间件的研究与对比[9], 最后选用了 HBuilder、HTML5+和 MUI 全套

开发工具进行 Hybrid App 开发[10]. 这里以“GMF 运动”应用为例子, 探讨如何以纯 Web 技术和中间件技术开发一个完整的 Hybrid 应用.

首先, 通过 HBuilder 新建移动 Web 应用开发框架, 得到 manifest.JSon, 然后进行 App 的全局控制. 在 manifest.JSon 文件中可配置应用信息、Appid、版本号、启动页、第三方登陆 sdk、地图定位 sdk、分享与埋点 sdk 以及各类 App 权限配置, 通过这里可以选择需要使用的选项和填写配置信息. 由于 GMF 运动的开发周期较长, 产品设计比较完善, 在配置文件中包括上述第三方 sdk 在内, 几乎所有的配置信息和密钥信息都已经配置成功和集成到 Hybrid App 里. “GMF 运动”App 中包含了 5 个主要模块, 分别是首页模块, 战队模块, 活动模块, 个人资料模块和订场模块, 而在各个模块中使用的 Web 技术和中间件技术都有重复的方面, 通过总结与分析, 除了常规的布局与业务功能开发外, 得出“GMF 运动”开发过程中以下 6 个开发的关键技术, 并通过“GMF 运动”具体页面进行分析.

2.1 HTML5+加载顺序

实际上, 对于 Web 文档的加载, 不管是从服务器上传递过来的, 还是缓存在本地的 Webview, 都需要先经过一定的加载顺序, 当 html 文档加载完渲染后会得到一棵 DOM 树, 代表 DOM 文档结构加载完成, 此时可以通过 JQuery 封装的$.ready() 来作为 DOM 树加载完成的分界. 当 CSS 文档下载完毕可以渲染成一棵 CSSOM 树, DOM 树与 CSSOM 树结合就会得到一棵 RENDER 树, 此时可以通过 window.onload() 来作为 RENDER 树加载完成的分界, 实际上 window.onload() 是在所有文档文件, 包括 JS 文件和图片文件下载完毕后就可以调用的函数, 所以可以作为 RENDER 树加载完成的标志. 而此时才是 HTML5+模块启动的时候, 可以通过 MUI 封装的 API 即 mui.plusReady() 来作为 5plus 模块启动的边界, 也可根据官方封装的 plusready 事件来进行判断, 如 document.addEventListner (“plusready”, plusreadyCallback, capture).

在“GMF 运动”中, 根据 plusready 事件作为模块启动的边界点, 然后进行关于 MUI 和 html5+的 API 的调用, 否则将会出现报错信息.

2.2 页面加载、跳转与架构的搭建

在当前纯 Web 技术构建的 App 里面, Webview 页面的架构和加载显得十分重要, 由于使用 Web 原生技术进行页面跳转和创建的性能逊于 Native 页面构建, 开发者需要通过 HTML5+封装原生窗口所提供的 API 来进行 Webview 的创建和跳转, 以实现性能的提升. 根据 5plus 封装的 Webview 窗口接口, 可以通过 mui.openWindow() 或 plus.Webview.open() 来进行窗口的创建和跳转, 当窗口的创建超过一定的时限, 就会调用原生模态层等待动画来解决白屏的问题. 为了性能的进一步提升, 开发者可以采取预加载的方法.

预加载即在用户尚未触发页面跳转的事件的时候, 在线程空闲的时候提前创建好对象页面, 这样在用户进行跳转操作的时候, 就可以立刻进行跳转, 无需进行页面和数据的加载, 极大的减少了用户等待的时间, 在性能低端的 Android 移动设备下体验优化更显著, 在 Hybrid App 性能优化中很重要, 而在页面的跳转过程中, 仅仅将页面进行显示和隐藏, 并不是创建和销毁. 具体实现方法与 5plus 提供的接口无关, 开发者可自定义函数方法在当前页面加载完成后对对象页面进行预加载操作, 值得注意的是由于页面加载完成是存储在内存中的, 所以必须对预加载页面进行统一的管理.

关于页面架构的搭建, 除了根据文件目录结构规范进行页面的构建和管理, 还有在下拉上拉刷新加载和首页框架的构建. 常见的 App 首页有导航栏与分页, 如用原生 Web 技术, 可以通过 div 的切换来构建, 这样窗口切换的速度将会得到提高, 但是文档的结构会变得臃肿, 所有页面都在同一个文档中构建会使 DOM 树和 RENDER 树的构建速度降低, 首页渲染的时间会受到影响, 因此需要进行窗口的管理.

对于 “GMF 运动”的首页, 根据结构可以创建主窗口 main.html, 内容只包含导航栏 nav bar 和页头 header, 内容填充则根据导航按钮数目建立相对应的子窗口, 加载于对应的位置, 点击相应的导航按钮则显示对应子窗口, 隐藏其他窗口, 则页面可根据顺序加载, 加快了首页的载入速度, 同时, 对应的页面逻辑也可以分开组织, 如图 3 所示.

对于刷新页而言, 大部分的 H5 框架都是使用 DIV 来模拟下拉回弹的动画, 在低端 Android 机型时常会出现卡顿的现象, 因此 MUI 采用了双 Webview 的技术来解决流畅度的问题, 在拉动的动作中被拉动的对象不是 DIV, 而是整个 Webview, 动画采用的是原生动画, 结构依旧采用的是主页面和子页面 Webview, 经过 MUI 的封装则可实现流畅的拖动动画.

图 3 首页框架逻辑结构

在“GMF 运动”的多处页面中都是用到双 Webview 刷新原生动画, 例如在战队模块中, 数据列表的众多数目需要使用下拉刷新进行数据请求, 即此处描述的原生刷新方案, 如图 4 所示进行页面构建.

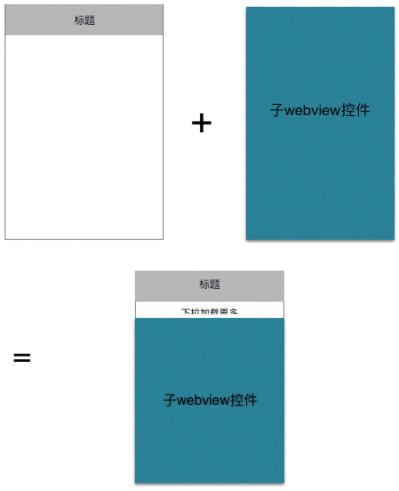

图 4 双 webview 下拉刷新原理

通过将两个 Webview 的组合, 在下拉时拖动子 Webview 页面, 页面的拖动采取的原生动画, 保证了拉动的流畅性.

2.3 页面间传值与事件触发

由于页面的创建在上述的分析里面分为即时创建和预加载创建.

在打开对象页面的过程中即时创建为创建并传值, 可以通过 mui.openWindow({url:(), id:(), extras:{//自定义扩展参数, 可以用来处理页面间传值}}) 这个方法, 与原生的窗口创建和页面间传值原理类似, 但是窗口创建速度过慢.

预加载页面则是从隐藏转为显示并传值, 机制上不一样. 将已经加载好的页面进行显示, 无法利用额外参数进行传值, 只能通过触发其他窗口的自定义事件并传值来实现. 具体实现方法为在子页面新建自定义事件并绑定到 window 对象上面: window.addEventListener(“customEvent”, function() {});

然后再通过 MUI 封装的 fire 接口以触发目标窗口的自定义事件: mui.fire(target, event, data), 通过在 data 对象中放入需要传送的值, 即可实现页面间传值.

另外上述方法不仅仅可应用于预加载页面间传值, 也可以实现不同页面间的数值更新, 事件触发. 如在“GMF 运动”中的战队模块图 (图 5) 中.

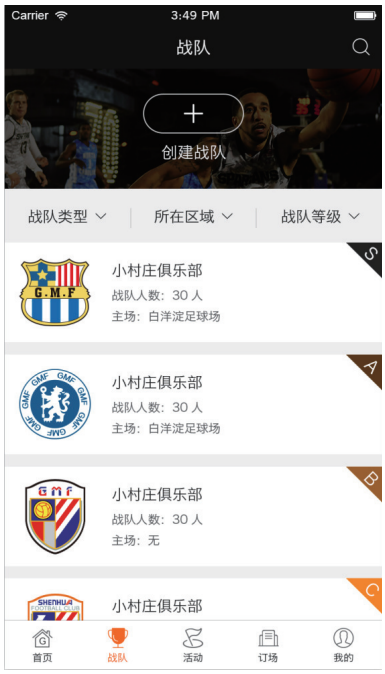

图 5 “GMF 运动”战队模块

如图 5 所示, 战队模块图中数据列表采用的是子 Webview 的形式, 上方的筛选按钮位于父 Webview 上, 根据 Webview 嵌套的原理, 子 Webview 的显示区域要处于父 Webview 之上, 无法通过控制 z-index 的形式以改变可视区域的渲染, 因此, 弹出框位于子 Webview 之上的需求将无法实现.

此时可以通过将弹出框置于子 Webview 上, 筛选框位于父 Webview 上, 通过对筛选按钮的监听, 触发子 Webview 的跨页面自定义事件, 实现对子 Webview 的 DOM 元素的显示与隐藏的控制, 也能通过值得传递更新子 Webview 页面中的数据列表.

2.4 第三方 SDK 的嵌入

在一开始介绍的 manifest.JSon 可以填入我们需要加载的第三方 SDK 的参数, 如 Appid, Appsercet, key 值等, 有相当一部分的第三方 SDK 已经集成在 5plus 的 API 里面, 只需要根据流程调用封装好的 API, 即可成功, 已知的封装好的 SDK 有微信支付、支付宝支付、第三方登录模块、个推推送、第三方分享、友盟数据统计. 如图 6 所示.

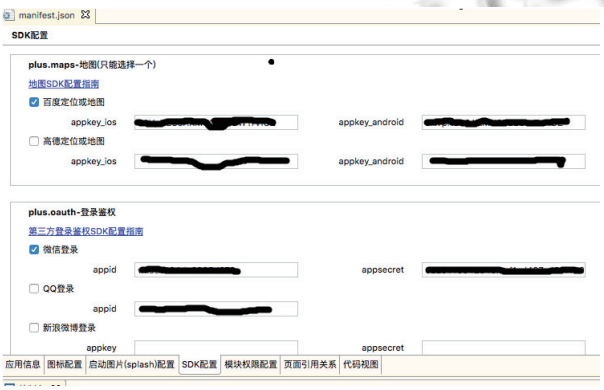

图 6 Manifest 文件配置第三方 SDK 图示

而未进行封装的 SDK 则需要通过 Web 技术来调用, 由于 Native 移动应用调用第三方 SDK 只能通过对应的语言来调用, ios 需要使用 objective-c 或者 swift, Android 需要用 java, 所以未封装的 sdk 尤其是没有进行 Web 版开发的 sdk, 需要耗费开发人员更多的学习成本, 没有进行 Web 版开发的 sdk 则不可调用. 然而大部分的第三方 sdk 都有网页版 sdk, 因此开发者可以通过 Web 技术进行调用, 通过填写对应的参数, 调用对应的 restful API, 根据流程向第三方服务器发送请求, 获取相关功能.

2.5 打包分发

在 Hybrid App 中, 尤其是以纯 Web 技术开发的 Hybrid App 中, 打包的过程没有办法像 iOS 那样使用 xcode 打包, 也没有办法像 Android 那样使用 eclipse 打包, 尤其是打包还需要配置众多参数. Hbuilder 提供了这样的打包功能, 只需要在 manifest.JSon 里面填写完整的版本管理信息, 配置好 Appid、urlschemes、启动图标、模块加载以及各类三方 Appid, 即可一键上传到云服务器进行打包, 上传之前需要按照 App 打包所需的证书和密码的填写, 方可正确打包. 上传到云服务器之后, 等待打包过程结束即可获得安装包. 获得安装包之后即可根据业务需要上传到 iTune Store、应用宝、360 市场等各类分发渠道进行分发了. 如图 7 所示.

图 7 一键打包封装参数界面

2.6 优化策略

一直以来, HTML5 的体验都被诟病, 在 iOS 的表现相对流畅, 但是在 Android 的低端设备上则表现卡顿, 与 Native App 对比性能非常差, 在 5plus 框架出现之后, 通过封装大量原生动画和组件, 使得以 5plus 和 MUI 为框架开发的 Hybrid App 有着出色的性能表现.

例如上述的拉动刷新加载动画调用的就是原生动画, 拖动的是整个 Webview 而非 DIV, 性能比 DIV 模拟动画要更好. 模态选择框、等待框、对话框等都已封装在 5plus 的 NativeUI 里面, 通过调用可以快速唤起原生 UI.

然而在创建 Webview 的过程中, 依然存在着较长的等待时间, 文档的加载和渲染也存在着延时, 请求的访问也存在着延时, 这一系列的体验问题仍然需要解决.

对于 Webview 的创建, 可以使用预加载的方法来避免创建时间过长, 当用户点击的时候即时显示, 唯一需要注意页面间传值的问题. 对于文档的加载和渲染, 网络的延时可以从多个方面着手, 由于 JS 的加载是阻塞型和单线程的, 所以需要合理规划资源的加载顺序和逻辑的执行顺序.

首先 css 的加载是异步的, 可以置于 html 的 head 标签内, JS 的加载由于是阻塞型的, 可以置于文档底部, 并且尽量缩小 css 和 JS 的大小, 剔除重复的逻辑.

尽量避免页面的回流和重绘. Opera 列出了一文: “reflow 和 repaint 是减缓 javascript 的三大主要原因之一”, 关于重绘, 当元素的外观的可见性 visibility 发生改变的时候, 就会引起重绘的出现, 例如元素的 outline、visibility 或 background-color 等属性, 在浏览器必须验证 DOM 元素可见性的过程中, 代价非常高昂. 关于回流, 当页面 DOM 元素结构发生改变, 页面布局发生改变的时候, 即引发回流, 而回流必引发重绘, 所以回流的代价更加高, 涉及到部分页面甚至整个页面的布局, 其中一个元素的回流导致了所有的子元素和及 DOM 中祖先元素其后的回流, 使得性能消耗非常大, 一般来说, 调整窗口的大小、改变了字体的大小、增加或者移除样式表、内容值的变化、激活了伪类的状态、使用 script 操作 DOM、计算 offsetWidth 和 offsetHeight 属性、操作类的属性等. 有可能大部分情况下我们没有办法避免去操作 DOM 元素, 没法避免重绘和回流, 但是大部分情况下为了达成同一个目的, 可以设法避免回流, 只触发重绘, 或者尽量减少回流的连带效应.

因此为了使回流和重绘对性能影响降到最低, 我们可以通过多种方法进行优化, 如尽量在 DOM 树的末端节点处改变样式, 动画效果尽量应用到定位属性为 absolute 或者 fixed 的元素上面, 这样不会影响其他元素的布局, 只会导致重绘而不会导致一个完整的回流, 降低消耗. 如避免使用 table 布局, 由于 table 布局内单元格元素的宽高发生改变会导致所在行和列的宽高发生改变, 因此一些细微的变化都会导致表格中其他的元素节点产生回流.

然后是后台接口与数据库的开发, 我们需要安装 PHP 环境 WAMP(即 Windows 环境下的 Apache 服务器, MySql 数据库和 PHP 开发环境), 这样就具备了基本的服务器开发环境条件了.

我们还需要安装 Codeigniter 这款 PHP 开发环境. Codeigniter 作为一款 PHP MVC 框架, 提供了足够的自由支持, 允许开发者更加迅速的进行开发工作, 其入门快速, 文档齐全, 还非常轻量, 是一款非常优秀的框架.

在 Codeigniter 安装成功后, 我们还需要在 github 上下载 Codeigniter 所支持的 API 接口包, 解压到 Codeigniter 框架对应的位置里面.

接下来就可以开始在 Codeigniter 里面修改对应的控制器和模型, 并开发出需要的 API 接口了, 这样开发好接口后, 我们便可以在移动端端使用 JS 脚本来进行接口的调用, 轻松远程上传和获取数据了.

如图 8, 我们在以上的开发过程中连接 iPhone7 实时真机调试运行了这个移动端的 GMF 运动平台了, 接下来我们就可以根据需求, 打包成各个平台所需要的运行安装文件了, 这是 Hbuilder 的优势, 融合了 Web 项目的跨平台特性, 可以轻松打包成相应平台的安装文件, 实现一份代码多处使用的目标.

图 8 运行中的 GMF 运动平台

3 实验结果和性能对比

对于 GMF 运动的实现进行性能分析, 需要从多个设备与环境进行分析, 分析的性能指标从启动时间, 页面渲染, 性能稳定、网络性能等进行评估.

本次分析将从以下条件进行分析.

1) 硬件环境支持

IOS: iPhone6/ iPhone6s/iPhone7 Plus;

Android: meizu MX3/ samsung s4/xiaomi 5s;

屏幕分辨率支持: 800*480 960*540 960*640 1280*720 1920*1080.

2) 软件环境支持

IOS: IOS7.0 或者更高版本;

Android: Android 4.0 或更高版本.

3) 性能指标

启动时间: 正常 App 启动需要保持在 5 秒内;

页面渲染速度: 应当保持在 1 秒内;

网络性能应当支持 2G, 3G, 4G 网络和 wifi 网络, 在网络信号不稳定、网络连接被重置时, 应当无闪退、卡顿、崩溃、黑白屏和内存泄漏等问题.

4) 测试方式

chrome 开发者工具;

Android 开发者模式;

植入计时统计代码;

对同一设备同一环境重复测试 5 次, 取平均值.

3.1 启动时间

测试首页加载时间, 通过设置启动页遮盖时间, 当 DOM 元素完全加载, 数据完全请求完毕后, 页面元素完全渲染成功后, 统计首页加载时间如下:

以 iPhone6、iPhone6s、iPhone6 plus、iPhone7 plus 等 iOS 设备进行测试;

以 Android xiaomi 5s、Meizu MX3、Samsung S4 等 Android 设备进行测试.

得出加载时间表如表 1 所示.

表 1 页面启动时间

页面启动时间	iPhone6	iPhone6s	iPhone6 plus	iPhone7 plus	Meizu MX3	Samsung GT19503	xiaomi 5s	Oppo r9s
第一次	1.9	1.3	1.6	1.1	4.6	4.2	3.3	3.6
第二次	2	1.4	1.4	0.9	5	4.3	3.2	3.9
第三次	2.3	1.3	1.8	1.2	4.9	4.5	3	4
第四次	2.2	1.3	1.6	1.1	5.1	4.3	3.1	3.9
第五次	2	1.4	1.5	1.0	4.7	4.2	3.3	4
平均	2.08	1.34	1.58	1.06	4.86	4.3	3.18	3.92

可以看出在测试过程中, 启动时间都在 App 标准启动时间 5 s 内, 但 iPhone 机型普遍表现比 Android 机型好, 加载速度都很优秀, 这是 iOS 平台对 html5 支持较好的体现.

3.2 页面加载时间

选取战队详情页和报名人数页, 如图 9.

图 9 中战队详情页为预加载页面, 报名人数页为即时加载页面, 因为预加载页面已无需创建 Webview 的多余时间, 在加载速度上预加载页面理论上要高出即时加载页面数倍时间, 对页面的加载进行计时代码植入, 收集数据并进行如下数据分析.

在父页面打开页面的前一段进行计时记录:

```
var beginTime = new Date();
```

在子页面数据异步加载完毕进行计时记录:

```
var endTime = new Date();
```

从而获得渲染时间差值:

```
var time = endTime.getTime() - beginTime.getTime();
```

表 2 和表 3 为数据表.

从以上两个页面的重复统计数据可以看出, 预加载页面的性能比普通即时加载页面的性能得到了大幅的提升, iOS 平台的加载性能依然比 Android 平台的性能要高. 在闲时预加载的性能提高方案在测试过程非常有效. 并且在所有页面和所有设备上, 加载平均时间都在 1 s 内, 属于正常的用户体验时间.

另外, 通过在多个网络环境进行多个设备的测试, 以中间件技术开发的 GMF 运动无闪退现象, 运行相对稳定.

战队详情页 报名人数页

图 9 加载页面图示

但是在部分页面上仍然会有白屏的出现, 部分控件仍有失灵的现象, 可以看出 Hybrid App 的开发依然还有改进的空间, 但从表现上和开发的成本上考虑, 以上述的开发模式进行移动应用的开发给予开发人员更多的选择空间, 给予企业更灵活的试运行产品的机会.

表 2 iphone 平台下四个页面的加载时间 (ms)

战队详情页加载时间	iPhone6	iPhone6s	iPhone6 plus	iPhone7 plus	报名人数页加载时间	iPhone6	iPhone6	iPhone6 plus	iPhone7 plus
第一次	56	50	60	20	第一次	135	120	134	80
第二次	30	46	45	13	第二次	146	145	99	94
第三次	45	33	34	25	第三次	167	180	102	92
第四次	60	50	35	12	第四次	201	110	133	87
第五次	50	65	49	30	第五次	160	159	150	84
平均加载时间	43.2	48.6	44.6	20	平均加载时间	161.8	142.8	123.6	87.4

表 3 Android 的加载据时间 (ms)

战队详情页加载时间	Meizu MX3	Samsung GT19508	xiaomi 5s	Oppo r9s	报名人数页加载时间	Meizu MX3	Samsung GT19508	xiaomi Ss	Oppo rds
第一次	89	127	56	79	第一次	370	890	122	189
第二次	97	130	69	90	第二次	401	670	210	253
第三次	101	140	65	66	第三次	332	987	180	280
第四次	86	133	77	63	第四次	369	1011	211	301
第五次	121	139	59	76	第五次	460	734	201	190
平均加载时间	98.8	133.8	65.2	74.8	平均加载时间	390.4	858.4	184.8	242.6

4 结束语

总结上文, 介绍了以 Web 技术进行 Hybrid 移动应用开发的中间件关键技术, 以 HTML5plus 和 MUI 作为开发框架进行 Hybrid 应用开发的实现细节和关键技术, 通过对这样的开发方式的理解和应用, 最后实现了“GMF 运动”应用平台的开发.

这样的开发方式使得移动端开发学习成本降低, 开发成本也进一步降低, 移动端开发人员的领域也开始转移, 但这类框架仍然有其局限和不足之处:

(1) 这样的开发方式不适合一直长期开发, 尽管优化方面已经非常接近原生开发, 但是仍然存在着体验上的差距, 对于对追求极致用户体验的开发者而言, 在以 Hybrid 开发作为快速开发抢占市场之后, 仍然需要开发原生应用, 在合适的场景使用 Webview 才是可取的方式.

(2) 这样的开发方式不适合在需要引入众多三方 SDK 的移动应用中使用, 目前 Hybrid 开发框架都会封装主流的第三方 SDK, 如支付、登陆、分享、IM 等. 但对于一些比较小众的第三方 SDK, 当 Hybrid 开发框架没有提供封装好的接口, 也没有提供 Web 版的 HTTP 接口, 则无法进行调用.

尽管如此, 对于绝大多数希望快速抢占市场的开发者而言, 使用中间件框架进行 Hybrid 应用开发仍然是最佳的选择, 尤其适合于重 UI, 重交互的移动应用开发. 使用 Web 技术进行的开发方式为移动应用开发注入了新的生命了, 使得开发的门槛和成本降低, 相信在 Web 技术不断的发展和优化下, 这样的技术和产品将会有更大的发展前景.

参考文献

1 Tian L, Du HC, Xu Y. The discussion of cross-platform mobile application based on Phonegap. Proc. of the 4th IEEE International Conference on Software Engineering and Service Science. Beijing. 2013. 652–655.

2 邹轩. 移动互联网跨平台应用中间件的研究[硕士学位论文]. 上海: 复旦大学, 2013.

3 唐红武. 基于 Native+html5 移动应用框架的研究. 计算机光盘软件与应用, 2013, (19): 297–298.

4 方锐. 基于 APPCAN 平台的 Android 手机电商客户端的设计与实现. 电子商务, 2012, (11): 54, 56.

5 Lee SH, Kim YH, Lee JK, *et al*. Hybrid app security protocol for high speed mobile communication. The Journal of Supercomputing, 2016, 72(5): 1715–1739. [doi: 10.1007/s11227-014-1318-3]

6 冯明. 基于混合模式 (Hybrid App) 移动终端设计的方法. 数字技术与应用, 2015, (4): 148–149, 151.

7 连允庆. 基于 HTML5 移动应用框架的研究及应用[硕士学位论文]. 成都: 电子科技大学, 2014.

8 吕昕. 基于 Web 的富客户端跨平台移动应用开发技术研究[硕士学位论文]. 昆明: 云南大学, 2015.

9 Wargo J M. PhoneGap essentials: building cross-platform mobile apps. Pearson Schweiz Ag, 2012.

10 常宁. 基于 HTML5 的房产信息采集系统的设计与实现[硕士学位论文]. 青岛: 中国海洋大学, 2015.