项目编号：	
密级：	绝密 机密 限制 一般
草稿文档：	
正式文档：	
正式文档修正：	

HP/Palm webOS 系统分析

项目名称： 盛大云计算前端系统

项目支持部门： 盛大创新院

作者姓名： 吕磊

盛大网络创新院

文档修改记录

序号	版本	修改者	主要修改内容	日期
1	V1.0	吕磊	初稿	2012.4
2	V1.0	潘爱民	修改	2012.5
3				
4				
5				

目录

1. 前言

下表是 HP webOS 平台的发展历程：

2007 年	硬件工程师 Jon Rubinstein 入职
2009.01	Jon Rubinstein 在 CES 展发布 Palm Pre
2009.01	Palm Pre 被投资界看好，称其为”iPhone 杀手“
2009.04	Palm Pre 上市
2009.06	Sprint 推出，实现跟 iTunes 同步
2009 夏季	Palm 跟苹果就 iTunes 同步发生纠纷，Palm 失败告终
2010.02	Palm 手机开始滞销
2010.04	HP 以 12 亿美元收购 Palm
2011.05	HP 推出首款 Palm 平台智能手机 Veer
2011.07	TouchPad 推出，但未获成功
2011.08	HP 宣布放弃所有 webOS 设备的运营
2011.12	HP 宣布将开源 webOS
2012 以后	各种收购传闻

表 1 HP webOS 平台的发展历程

webOS 的设计理念是非常大胆、非常前卫的，其面世之初也曾被广泛看好，但市场反馈并不理想，为什么呢？ 本文将以 webOS 3.0.4/3.0.5 为对象，从技术角度全面剖析 webOS。

2. 简介

初拿到 TouchPad 时，心理的热乎劲一闪则过，为啥？启动太慢了！下面是笔者用秒表计算的启动时间：

开机时间	1 分 16 秒
重启时间	1 分 38 秒
关机时间	22 秒

表 2 HP webOS 启动时间

1 分 16 秒的开机时间是有点过了，众所周知当前一些 Android 系统的启动时间号称在 30 秒以内，尤其是 TouchPad 的配置并不差，下表是 TouchPad 的硬件配置：

配置项	配置说明
CPU	Snapdragon 双核，APQ8060，1.2GHz
GPU	高通 Adreno220，45nm，支持 OpenGL ES
内存	1G
屏幕尺寸	9.7 英寸
分辨率	1024*768
音频	内置相关芯片，支持 MP3，AAC，AAC+，eAAC+，AMR，WAV，MID
视频	内置相关芯片，支持 MP4，3GP，AVC，AVI，MPEG-4 格式
电池	锂电池，6300 毫安
摄像头	130 万象素
传感器	GPS、重力感应、环境光线感应、距离感应器

表 3 HP webOS 硬件配置

1 分多钟的启动时间在这样的配置条件下只能说明系统启动部分没有充分优化好。下面列举了 webOS 运行时其他参数，包括内核信息、内存占有量、进程数等。

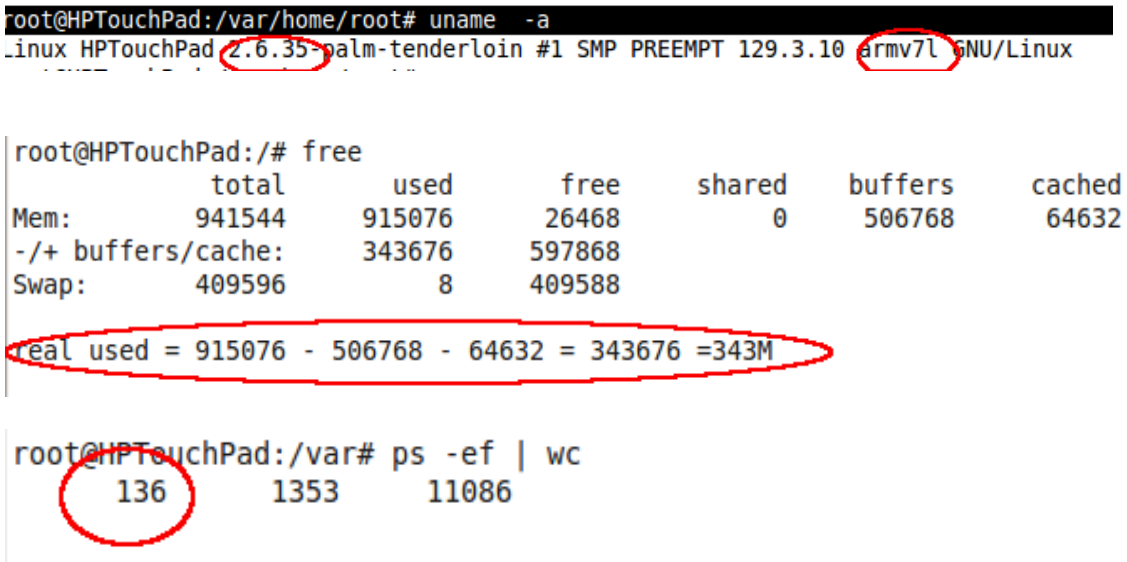


图 1 HP webOS 内存消耗

343M 的内存占有量勉强在可接受的范围内，但整体来看，HP 的 webOS 系统并不理想，同一个精雕细琢的产品相比仍存在一定差距。下面将从其 IPC、Services、GUI、Framework 等多个方面来全面地分析它。

3. HP webOS 架构

首先看一下 webOS 的体系架构，如下图 2 所示。webOS 底层是 Linux 内核，通过一个跨进程通信的设施，Service Bus，将各个进程，包括 App 进程和服务进程，连接起来。webOS 支持 Web 的应用和本地应用，Web 引擎使用了 WebKit，JavaScript 引擎使用了 V8。JS 服务可以基于 Node.js 来开发；系统本身提供了大量的服务，本文后面将会讲到。

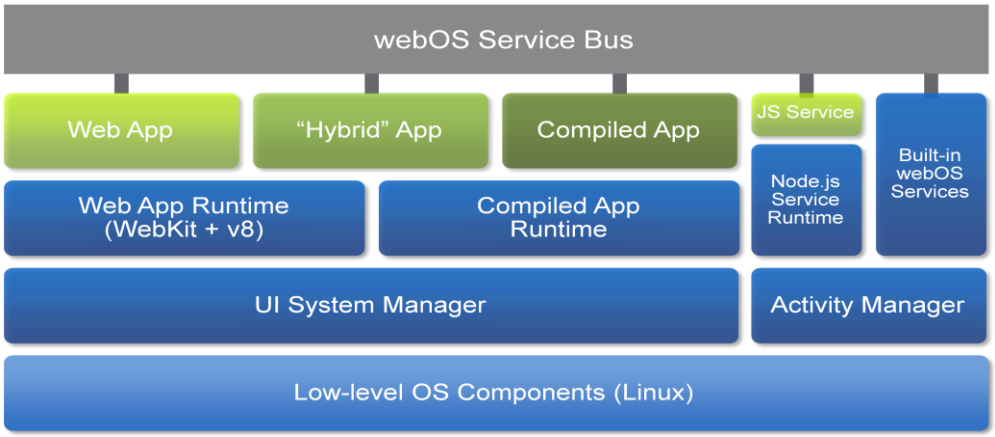


图 2 HP webOS 架构

4. 运行时组件

从运行时看，webOS 由 Web App、Native App、Hybrid App、Browser App、Services 构成，如下图 3 所示:

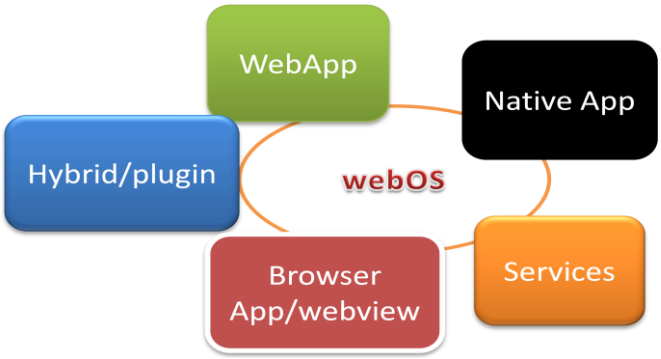


图 3 运行时组件

- Web App：使用 Web 技术开发，由 HTML/CSS/JavaScript 构成，一般使用 webOS 发布的 Mojo/Enyo 框架开发；
- Native App：主要使用 webOS 发布的 PDK 开发包开发，核心语言是 C/C++；
- Hybrid App：混合开发方式，即使用 Native 和 Web 联合开发，通过 plugin 交互；
- Browser App：跟 WebView 控件相关的 App，本质上可以归类于 Web App，但由于其实现特殊，这里单独列出来；

- Services：后台服务，包括 webOS 提供的系统服务以及用户开发的服务。

各类 App 如何运行，如何协作呢？

4.1. App 运行和协作

首先看看 WebApp 的运行。在 HP webOS 中，所有 WebApp 运行在同一进程 LunaSysMgr 里，如下图 4 所示：

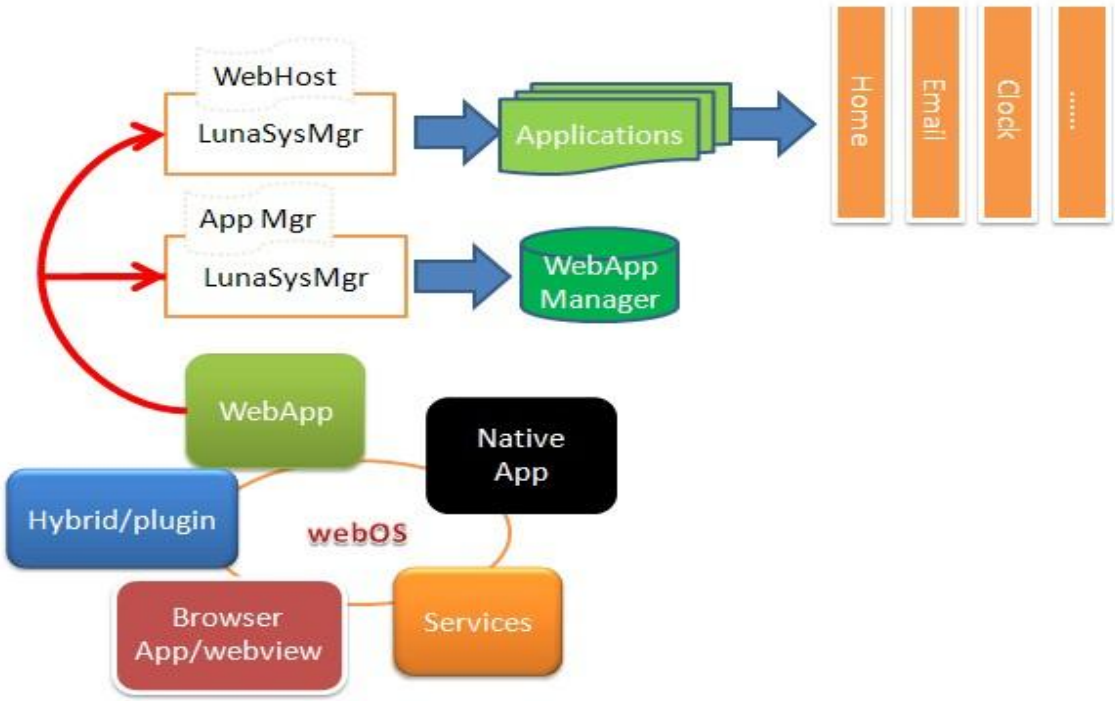


图 4 WebApp 运行图

所有 WebApp 运行在同一进程 LunaSysMgr 里，这是 webOS 的一个重大特征。LunaSysMgr 类似 X-Server，承担 input、display、window manager、WebApp manager、分发 IPC messages 的功能。所有 WebApp 运行在同一进程内，有利于窗口切换、资源管理，加速 WebApp 启动；但缺陷也是显而易见的，比如可能的内存碎片以及是否足够稳健以便支撑所有应用等。LunaSysMgr 运行时会被 fork 成两个进程，其中一个是 WebApp Manager，用于管理 Web App，接收来自其他进程的 command，如启动新的 WebApp 等；另一个为 Web App 真正运行的进程，主要利用 WebKit 浏览器引擎解析 WebApp 并渲染之。

WebApp 通常需要跟其他 WebApp 或 Services 协作来完成某一共同功能，由于 WebApp 的主要开发语言是 JavaScript，因此如何在 JavaScript 中访问本地服务呢？这通过 JavaScript 扩展对象 PalmServiceBridge 完成。PalmServiceBridge 对象是 Palm 自己的扩展，是 Window 对象的子对象，通过修改 WebKit 代码实现，其主要提供 onservicecallback 属性供 JavaScript 访问。

PalmServiceBridge 对象在 webOS 1.0 中采用 Applet 实现，尽管这完全兼容了 Web，但 Applet 运行时需要 JVM，这对于嵌入式系统来说，同时存在 JS 引擎和 Java 引擎，显然有点重了，性能是个大问题。因此在 2.0 以后，webOS 放弃了 JVM，而通过修改 WebKit 代码来实现 PalmServiceBridge 对象。

接下来看看 Native App 的运行机理。Native App 运行在单独进程中，如下图 5 所示：

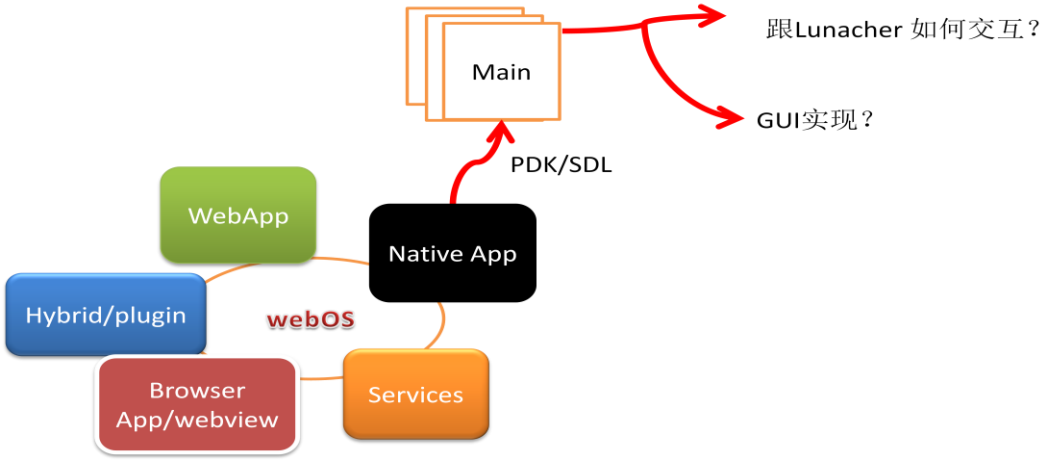


图 5 Native App 运行逻辑图

其如何跟 LunaSysMgr 进程交互并被管理呢？这要从 HP webOS 的 Native App 开发包 Palm PDK 说起。出于性能考虑，webOS 支持原生 Native 程序，这就是 Palm PDK 的功能，其主要目标是使得 webOS 支持原生的 OpenGL（ES）3D 游戏。Palm PDK 通过 libSDL 库提供相关支持。为实现 Native App 程序跟 LunaSysMgr 的融合，Palm 给 LibSDL 打了 patch，主要做了如下修改：

- 增加 Napp Event Thread，用于处理跟 LunaSysMgr 通信
- 不是将 LibSDL 的 draw 函数改用 opengles 实现；实际逻辑是：用户使用 opengles 的 api 来编写程序，但 opengles 所需要的 surface 以及 surface 的管理由 libSDL 提供
  - 调用 eglCreateWindowSurface 创建 surface，这里并不使用 UMP 内存，也不需要将 libSDL 的 surface 中的 pixels 直接赋值给 eglCreateWindowSurface 中的 pixel，可以理解为 webOS 的 libSDL 实质上抛弃了自己的 surface 相关管理，只是用于保存相关的 graphics 信息
  - 实现 FBCon_GLES_SwapBuffers 程序，供应用程序调用
- 相关代码
  - 1. SDL_fbgles.c
  - 2. events/webOS/*，App 事件处理相关代码

修改后的 libSDL 程序（Native App）架构如下图 6 所示：

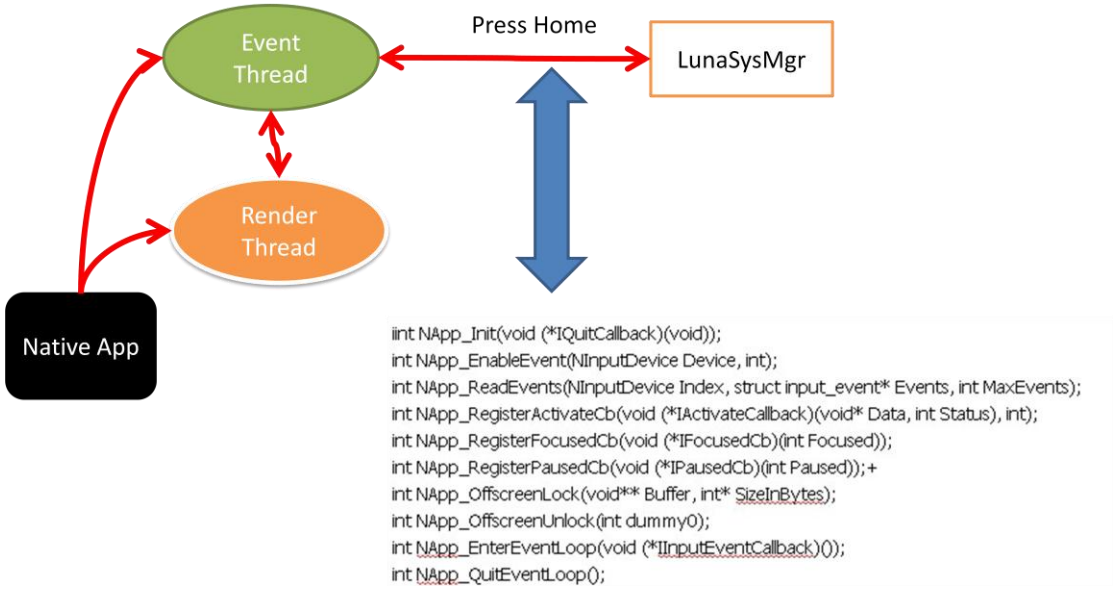


图 6 Native App 交互

由上图知，通过在 Native App 的 Event Thread 线程中处理来自 LunaSysMgr 的消息来完成跟系统的整合，例如处理 home 按键消息。关于 Palm PDK 以及具体 GUI 交互在后续还有介绍。

接下来看看 Hybrid App 的运行情况。Hybrid App 顾名思义，是在同一个 App 里集成了 Native 代码和 Web 代码，其运行机理如下：

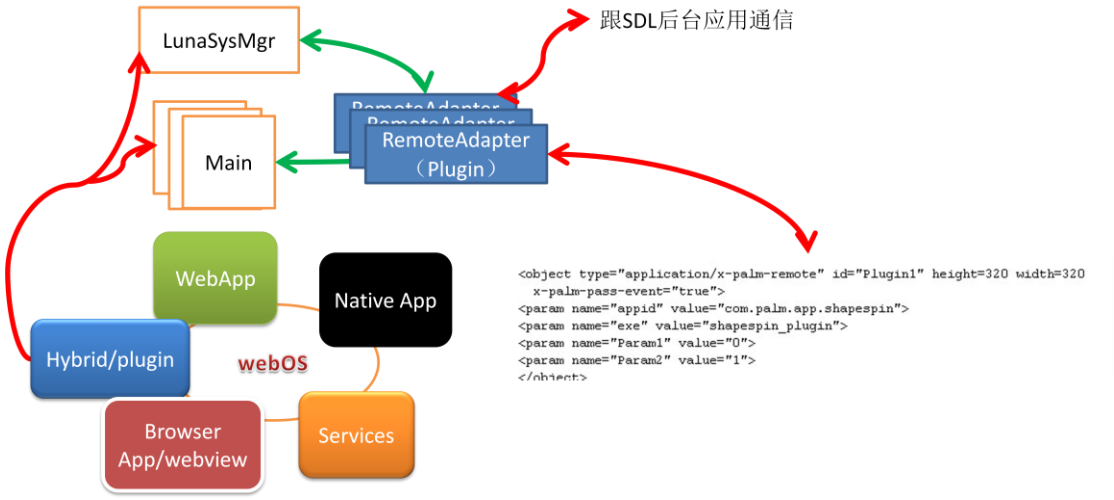


图 7 Hybrid App 运行逻辑图

由上图知，Hybrid App 的 Web 代码运行在 LunaSysMgr 进程中，Native 代码作为一个单独的进程存在，它们之间的通信通过插件 RemoteAdapter 完成。细节可参考 6.2 节的例子。

接下来看看 Browser App 的运行机理。Browser App 是指使用了 WebView 控件的 Web 应用，尽管它们可以看作是 Web App 应用，但由于其实现存在特殊之处，所以单列出来。WebView 控件可以看做 WebKit 浏览器引擎的一个实例，利用该控件你可以打开普通网页。运行结构如下图：

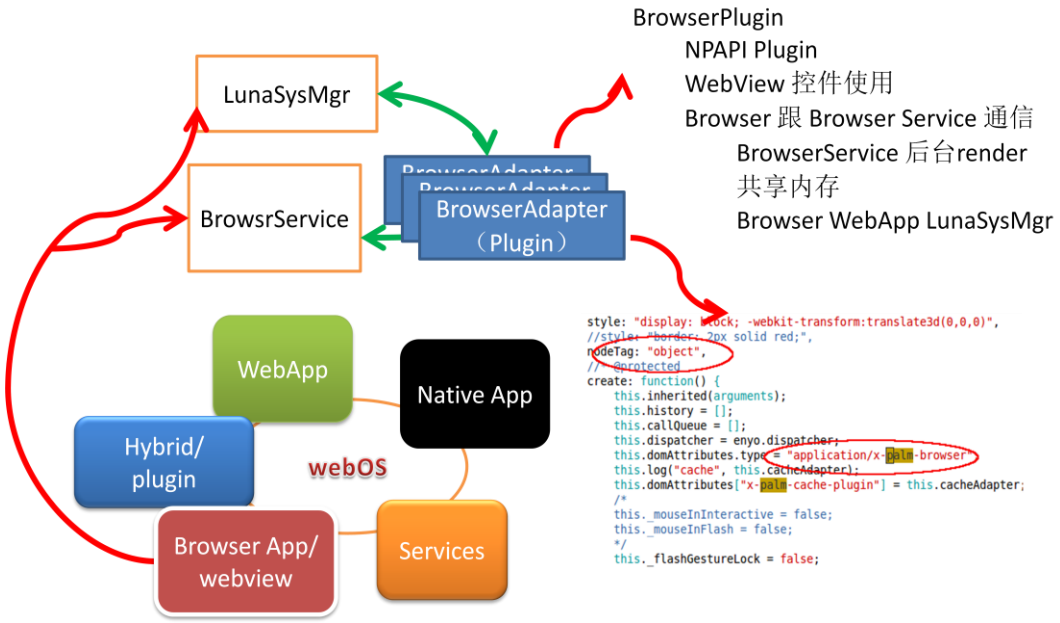


图 8 Browser App 运行逻辑图

由上图可知，Browser App 运行在两个进程中：LunaSysMgr 和 BrowserService。LunaSysMgr 运行 Web 相关代码，其中还包括一个插件 BrowserAdapter。BrowserService 是浏览器服务，用于解析、渲染网页，渲染后的内容通过插件 BrowserAdapter 被 LunaSysMgr 获取并显示。其渲染内容的交互参见 GUI 章节。

最后看看 webOS Services。webOS 提供了很多后台服务以利于开发者实现一些复杂的功能，包括电源管理服务、蓝牙服务、音频服务等等，下图是其服务运行图：

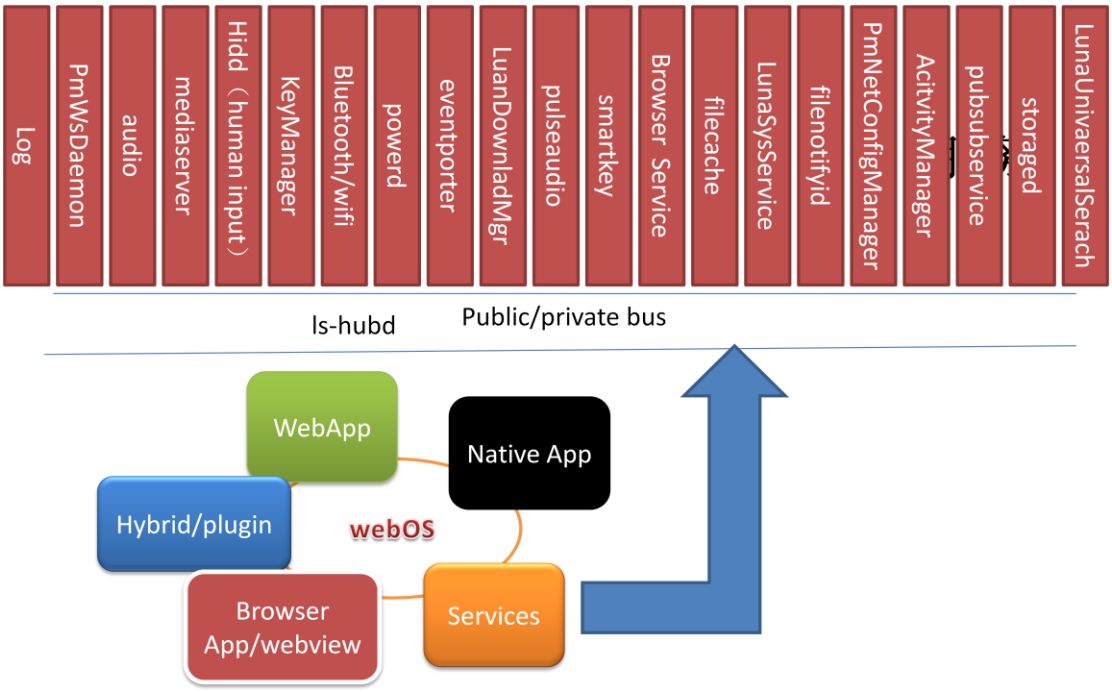
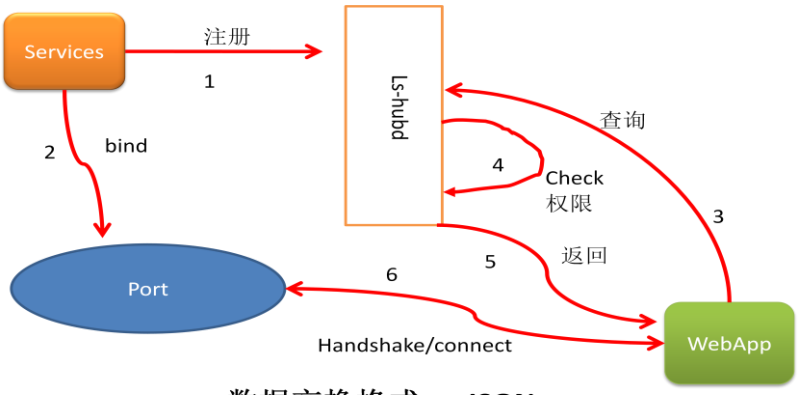


图 9 Services

所有服务运行的基础是 bus 的实现，通过 ls-hubd 程序提供，bus 即总线，是进程间通

信的场所，webOS 1.0 中采用第三方库 dbus 实现；2.0 及后续版本放弃了 dbus，并仿照 dbus 的逻辑实现了一套自己的 IPC 机制，其核心是采用 UNIX domain socket 来完成进程通信功能。下面是运行逻辑图：



数据交换格式 == JSON

图 10 Service 交互

首先某个 Service 通过向 bus 注册其提供的服务，同时 bind port 用于提供服务；WebApp 可以通过 bus 查询服务，bus 完成权限检测后返回结果；WebApp 根据返回的结果跟 service 建立连接并完成通信。可用 webOS 提供的 ls-monitor 查看当前提供的所有服务，如下：

```
PRIVATE HUB CLIENTS:
PID         SERVICE NAME                    EXE                                TYPE                 UNIQUE NAME
1596        com.palm.vpn                    /usr/bin/PmVpnDaemon               dynamic              /var/run/ls2/CPB4Gg
1117        com.palm.smartKey               /usr/bin/SmartKeyService           static               /var/run/ls2/VMKNSO
1617        com.palm.storage                /usr/sbin/storaged                 dynamic              /var/run/ls2/WoTCsg
1301        (null)                          /usr/bin/LunaSysMgr                unknown/client only  /var/run/ls2/qtlsU4
1301        com.palm.vibrate                /usr/bin/LunaSysMgr                static               /var/run/ls2/HpZY5F
1084        com.palm.keymanager             /usr/bin/keymanager                static               /var/run/ls2/DQOYNG
1083        (null)                          /usr/bin/LunaDownloadMgr           unknown/client only  /var/run/ls2/6x3wec
1065        com.palm.firewall               /usr/bin/PmNetConfigManager        static               /var/run/ls2/C1xBH3
1301        (null)                          /usr/bin/LunaSysMgr                unknown/client only  /var/run/ls2/fypmLv
1280        com.palm.activitymanager.client /usr/bin/activitymanager           unknown/client only  /var/run/ls2/HSVhwY
1095        com.palm.db                     /usr/bin/mojodb-luna               static               /var/run/ls2/WPt4Ut
1118        com.palm.browserServer          /usr/bin/BrowserServer             static               /var/run/ls2/Jeknpj
1143        (null)                          /usr/palm/nodejs/node              unknown/client only  /var/run/ls2/tK2w9m
1693        com.palm.monitor                /usr/bin/ls-monitor                unknown/client only  /var/run/ls2/RVeevx
1315        com.palm.luna-1315-phone        /usr/bin/LunaSysMgr                unknown/client only  /var/run/ls2/Pi8dHR
1315        (null)                          /usr/bin/LunaSysMgr                unknown/client only  /var/run/ls2/f39zkW
1280        com.palm.activitymanager        /usr/bin/activitymanager           static               /var/run/ls2/TMGOpN
1085        com.palm.audio                  /usr/sbin/audiod                   static               /var/run/ls2/ppJvgB
1301        com.palm.nativealertmanager     /usr/bin/LunaSysMgr                static               /var/run/ls2/KTkdoI
1645        (null)                          /usr/palm/nodejs/node              unknown/client only  /var/run/ls2/Kv65v7
1315        (null)                          /usr/bin/LunaSysMgr                unknown/client only  /var/run/ls2/E50eo4
1104        com.palm.telephony              /usr/bin/PmWsfDaemon               static               /var/run/ls2/YADHoO
1613        (null)                          /usr/palm/nodejs/node              unknown/client only  /var/run/ls2/ooRFGB
1301        com.palm.eventreporter.LunaSysMgr   /usr/bin/LunaSysMgr                unknown/client only     /var/run/ls2/eX9
1104        com.palm.messagingrouter        /usr/bin/PmWsfDaemon               static               /var/run/ls2/KCBswe
1065        com.palm.connectionmanager      /usr/bin/PmNetConfigManager        static               /var/run/ls2/ajNcaz
1301        com.palm.keys                   /usr/bin/LunaSysMgr                static               /var/run/ls2/Voe5b9
1296        com.palm.pubsubservice          /usr/bin/pubsubservice             dynamic              /var/run/ls2/3YUKWj
1126        com.palm.power                  /usr/sbin/powerd                   static               /var/run/ls2/CzrOeY
1234        com.palm.systemservice          /usr/bin/LunaSysService            static               /var/run/ls2/FXVPJi
1315        com.palm.luna-1315              /usr/bin/LunaSysMgr                unknown/client only  /var/run/ls2/GILQti
1075        com.palm.hidd                   /usr/bin/hidd                      static               /var/run/ls2/5gF1u3
1645        com.palm.appcatalog             /usr/palm/nodejs/node              dynamic              /var/run/ls2/r71zkO
```

图 11 系统服务图

服务的核心是 bus 的实现者，即 ls-hubd 程序。ls-hub 中包含了各个服务配置选项、服务权限、接受服务注册等等。一个标准的服务配置文件可能如下：

```
[D-BUS Service]
Name=com.palm.mediad
Exec=/usr/bin/mediaserver
Type=static
```

图 12 Service 配置

服务权限文件可能如下：

```
[{
        "role": {
                "exeName":"js",
                "type": "regular",
                "allowedNames": ["com.palm.mediapermissions", "com.palm.filenotifyd.js"]
        },
        "permissions": [{
                "service":"com.palm.mediapermissions",
                "inbound":["*"],
                "outbound":["*"]
        },{
                "service":"com.palm.filenotifyd.js",
                "inbound":["*"],
                "outbound":["*"]
        }]
}
```

图 13 Service 权限

webOS 还提供了工具 luna-send 用于跟服务交互，例如可以用如下命令启动 webOS 的浏览器，

- luna-send -n 1 palm://com.palm.applicationManager/launch '{"id":"com.palm.app.browser"}'

笔者用 strace 工具跟踪了 ls-hubd 的启动以及运行 luna-send 跟 ls-hubd 的交互过程，如下图：

```
  statfs("/dev/shm", {f_type=0x1021994, f_bsize=4096, f_blocks=512,
  futex(0xb7e4a860, FUTEX_WAKE, 2147483647) = 0
  open("/dev/shm/sem.PmLogLib", O_RDWR|O_NOFOLLOW) = 3
open("/usr/share/dbus-1/system-services/com.palm.systemservice.service", O_RDONL
fstat64(4, {st_mode=S_IFREG|0644, st_size=100, ...}) = 0
read(4, "[D-BUS Service]\nName=com.palm.systemservice;com.palm.image\nExec=/usr/
read(4, "", 4096)                       = 0
close(4)                                = 0
write(2, "LUNASERVICE ERROR 4: Error opening directory '/var/mft/palm/ls2/roles/
  = 249
write(2, "Done parsing role directories\n", 30Done parsing role directories
) = 30
write(2, "Role: ref: 1, exe_path: \"/usr/bin/filecache\", type: 0, ", 55Role: re
write(2, "allowed_names: ", 15allowed_names: )          = 15
write(2, "com.palm.filecache ", 19com.palm.filecache )      = 19
write(2, "\n", 1
recv(21, "\1\0\0\0\24\0\0\0\23\0\0\0com.palm.telephony\0\0\0\0\0\0\0\0\0\0", 40,
recv(21, "+\0\0\0\5\0\0\0\r\0\0\0", 12, MSG_DONTWAIT) = 12
recv(21, "_private_service_status\0com.palm.telephony\0", 43, MSG_DONTWAIT) = 43
recv(21, ",\0\0\0\6\0\0\0\26\0\0\0", 12, MSG_DONTWAIT) = 12
recv(58, "\1\0\0\0\34\0\0\0\34\0\0\0com.palm.applicationManager\0\1\0\0\0\0\0\0\
recv(58, 0x80a9ee4, 12, MSG_DONTWAIT)   = -1 EAGAIN (Resource temporarily unavai
send(58, "h\0com.palm.applicationManager\0\1\/var/run/ls2/Xy6r8g\0\0\0\0\0\0\0",
sendmsg(58, {msg_name(0)=NULL, msg_iov(1)=[{"\0", 1}], msg_controllen=16, {cmsg_
close(66)                               = 0
```

config

permissions

service start

luna-send

图 14 Service 跟踪

其中 config 和 permissions 是 ls-hubd 启动时读取相关服务配置文件和权限文件的信息，service start 是服务启动时向 bus 注册的信息；luna-send 是使用 luna-send 命令时，总线（ls-hubd）收到的服务查询请求信息。所有的数据交换采用 json 格式。

HP webOS 目前提供的服务如下：

- PmWsDaemon，电话服务；
- Audio，音频服务；
- Power，电源管理服务；
- Mediaserver，多媒体服务；
- NetWork，网络模块服务；
  - Wifi；
- BlueTooth，蓝牙服务；
- Hidd—(Human input device daemon)；
  - GPS；
- Filecache，本地文件 cache 服务；
- Filenotifd；
- Keymanager（加/解密支持）；
- Browser service；
- Acitivity Manager；
- Storaged --db8，embedded database JSON support；
  - 跟 HTML5 indexedDB/localstorage 无关，是 webOS 自己的存储服务，用于满足 robust、high-performance 应用程序的需要；
  - 如 contact 数据的保存；
- Application Manager (open/lunach webapp)；
- LunaDownloadMgr；
- LunaUniversalSerach；
  - Just type；
- Pubsubservice；
  - Publish Subscribe Service。

HP webOS 的 Activity Manager 是个特殊的服务。在 Web App 开发中，经常需要跟其他 WebApp 或者服务交互协作来共同完成某一功能，Activity Manager 就被用于协助完成该功能。例如如下 demo：

图 15 Activity Manager

图中上面部分是 Web App 先注册其自己的 Service：com.palmdts.helloworld.service，并提供 hello 的方法；下面是 Web App 请求 Service 的代码，调用 this.controller.serviceRequest 请求 com.palmdts.helloworld.service 服务的 hello 方法，这里实质是创建了一个新的 Activity。关于 Activity Manager 和 Activity 功能大致如下：

- 交互协作的平台
  - WebApp 和 WebApp 之间的交互
  - WebApp 和 Services 之间的交互
  - 发布/订阅服务
  - 调度（队列）
- Activity

  从名字上看，Activity 借鉴了 Android 中 Activity 概念，用于各个 Application 之间相互协作完成某一功能，App 能请求 Activity manager 创建 Activity，并设置其运行条件；当条件满足时由 Activity Manager 统一调度运行。

HP webOS 提供的服务涵盖一个 OS 的各个方面，比较全面，也借鉴了 Android 的很多成功的概念，如 Activity。但其运行基础，即 IPC 的实现并无亮点，早期更是直接采用了 dubs---针对 X11 桌面的 IPC 机制；而且其数据交互格式 JSON 相对于二进制数据交互来说也是比较弱的。

从 HP webOS 整个运行情况看，PalmServiceBridge 从 Applet（JVM）变更为 WebKit 原生

扩展；放弃 dbus；以及针对 WebView 控件提供特定的 BrowserService 服务，等等，尽管无法获取其真实的意图，但无外乎性能相关，毕竟这是运行在终端设备上的系统。但从开发角度看，HP webOS 绝不是一个精雕细作的系统，而仅仅是为了 webOS 概念而生的系统，一个拼凑的系统。在系统层面缺乏细致的规划，譬如还有 Applet（JVM）的痕迹，dbus 的痕迹，这对于终端系统而言还是显得太重了。

5. GUI

一个好的系统，少不了一个好的 GUI 系统，Mac/Windows 莫不如此；尤其现在 GPU 飞速发展，如何更好地利用 GPU 功能来完成 2D/3D 绘制以提升系统性能是大多 GUI 系统的首要目标。HP webOS 在这方面也下了一定功夫。下图是 HP webOS 最核心的程序 LunaSysMgr 的 GUI Stack：

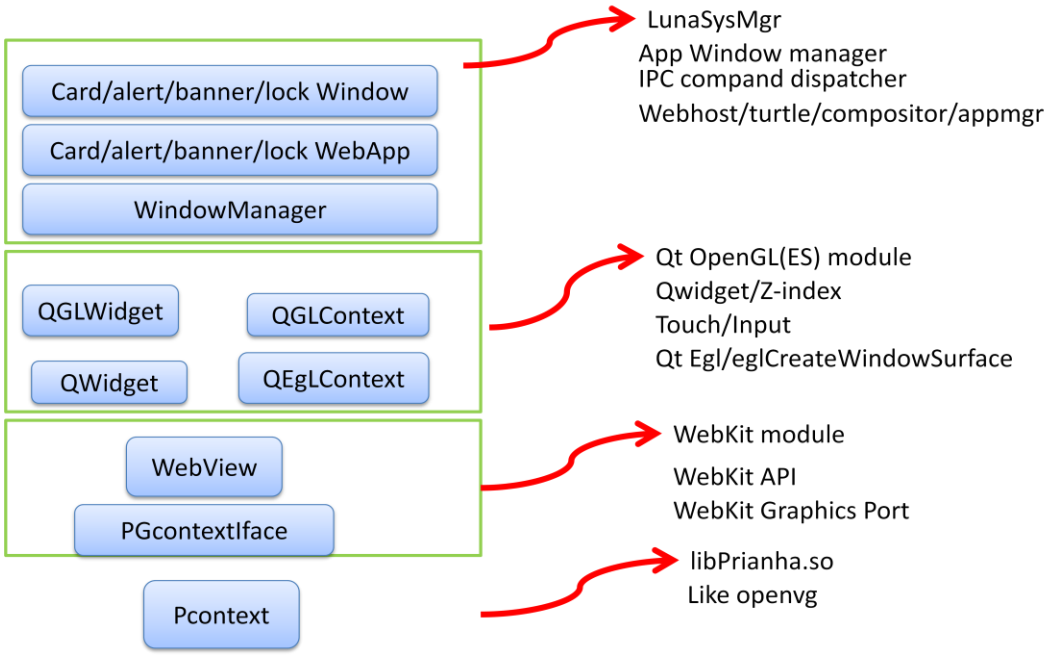


图 16 LunaSysMgr Stack

由上图知，HP webOS 使用了成熟的 GUI 系统 Qt，而且核心是使用了 Qt OpenGL(ES)模块；其浏览器引擎采用 WebKit，并对 WebKit Graphincs Port 做了一定的工作，主要为封装了 EGL 接口，这样确保在 WebKit 的 2D 渲染中能用上硬件加速；同时提供了一个 2D 矢量库 libPiranha.so，该库针对模拟器和实际硬件存在两个版本，可以理解为模拟器上采用软件实现 2D 绘制函数；实际硬件通过调用 opengl es 来实现 2D 绘制，即通常说的硬件加速，从功能上看 libPrianha.so 跟 OpenVG 库类似。为什么 HP webOS 不直接使用 OpenVG 而另开炉灶自己实现了 libPrianha.so 就不得而知了。但从 HP webOS 发布的 WebKit 代码来看，其在设计之初就考虑到使用硬件加速功能的，主要表现为：对比目前公开的 WebKit 两份代码，一份是 TouchPad 所使用的关于 WebKit 的 patch 部分，另一份开源的 WebKit 代码，这两份代码在其 Graphics 部分存在较大的差别，前者关于 graphics 使用的是 HP 自己的 pg 目录，后者使用的是 qt port。pg 目录跟后来 WebKit 为了支持 OpenVG 直接渲染而增加的代码基本类似，因此可以猜测 HP webOS 在促使使用硬件加速方面做了一定努力。下图是 libPrianha.so 库的一些符号信息：

```
00007e00 T PGLESContext2D::CacheVTrans()
000088a0 T PGLESContext2D::DrawPolygon(PVertex2D*, unsigned int, bool)
000095d0 T PGLESContext2D::SetClipPath(PGLESPath*, PFillRule)
00091b0 T PGLESContext2D::SetCompMode(PCompRule)
00007350 T PGLESContext2D::SetFillPick(PVertex2D, bool*)
000073a0 T PGLESContext2D::SetNoStroke()
000092e0 T PGLESContext2D::FreePrograms()
00008b90 T PGLESContext2D::FreeVertices(PVertex2D**, unsigned int, bool)
000073c0 T PGLESContext2D::SetAttribute(unsigned int, unsigned int)
00007200 T PGLESContext2D::SetFillColor(PColor32)
00007220 T PGLESContext2D::SetFillLGrad(PGLESColorRamp const*)
00007240 T PGLESContext2D::SetFillRGrad(PGLESColorRamp const*)
00007250 T PGLESContext2D::SetFillRGrad(PGLESColorRamp const*, PVertex2D, PVertex2D, PFixed)
00009f80 T PGLESContext2D::CreateProgram(PGLESContext2D::PFillStyle*)
0000fa70 T PGLESContext2D::DrawRectangle(PVertex2D, PVertex2D, PVertex2D)
0000c1d0 T PGLESContext2D::DrawSubPixmap(PGLESPixmap*, PVertex2D, PVertex2D, PVertex2D, PVertex2D)
0000b580 T PGLESContext2D::DrawTextAtlas(unsigned short*, unsigned int, PVertex2D, PVertex2D*)
0000f030 T PGLESContext2D::RasterPolygon(PVertex2D*, unsigned int, bool)
0000f910 T PGLESContext2D::RasterPolygon(PVertex2D**, unsigned int)
00008e40 T PGLESContext2D::SetClipRegion(PRect)
00007310 T PGLESContext2D::SetFillPixmap(PGLESPixmap const*)
00007320 T PGLESContext2D::SetStrokePick(PVertex2D, bool*)
000087c0 T PGLESContext2D::RasterPolyline(PVertex2D const*, unsigned int, bool)
00009da0 T PGLESContext2D::RenderToWindow(void*)
```

图 17 libPrianha.so 的符号信息（部分）

看到这里，读者可能会问，既然 HP 自己实现了其 2D 渲染库 libPrianha.so，那么 Qt 模块在 HP webOS 中的作用是什么呢？实质上，webOS 1.X 系列确实没有使用 Qt 库，2.0 以后为什么选择了 Qt 库到底是技术原因还是其他原因尚不清楚，猜测可能是基于一些特效的考虑，例如基于 Qt 的 opengles 模块的要求或者一些特效的考虑。言规正传，Qt 目前的作用主要包括 touch 的处理、window/z-index 的管理。

下面我们看看 HP webOS 的一些特效的实现。在 TouchPad 上，按下 home 键，当前窗口会缩小到屏幕中央，屏幕交给桌面程序处理；由上一部分的介绍可以知道，当前窗口可能是 WebApp，也可能是 NativeApp 的窗口。那么如何实现将当前窗口的最后一帧内容绘制到桌面应用里呢？

- 当前窗口是 WebApp 的窗口，由于桌面应用和 WebApp 都运行在 LunaSysMgr 进程里面，因此很容易获取到当前窗口的内容
- 当前窗口是 Native App 窗口，这时当前窗口和桌面分属不同进程，Native App 自身进程和 LunaSysMgr 进程
- 当前窗口 Browser App 窗口，实质上 Browser App 跟 Web app 类似，尽管存在另一个进程 Browser Service，但由于其显示在 LunaSysMgr 的 plugin 里面，因此也容易获取当前内容

综上所述，这里重点介绍下 Native App 的绘制过程，如下图：

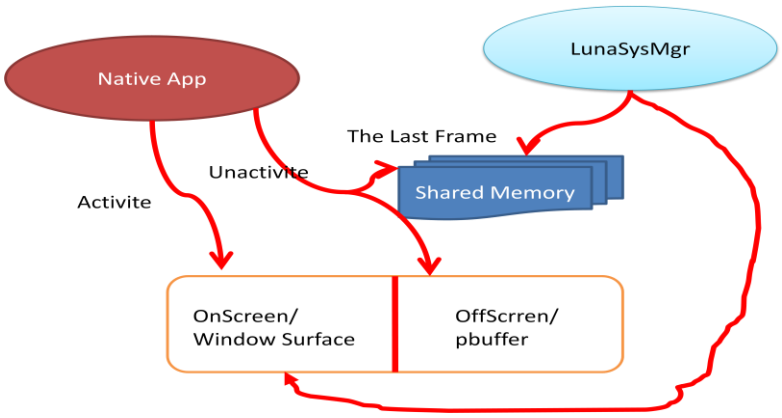


图 18 Naitve App 绘制

由图知，当 Native App 处于活跃状态时，它是直接渲染到屏幕的，此过程很高效；当它被切换时，如按下 Home 按键，它会将当前最后一帧内容 copy 到 shared memory 中，同时分配 pbuffer（或者 off Screen）供 Native App 使用；LunaSysMgr 使用共享内存显示在桌面上。至于当前 Native 进程被切换后是否再绘制（这里是 pbuffer）就取决于开发者的实现了。可见，对 Native App 来说，它最大化地利用了 GPU 图形功能，因此是相当高效的。

在 HP webOS 的应用中，存在一类特殊的应用即 Browser App。由于 Browser App 部分渲染的内容在另一进程 Browser Service 中，因此需要将 Browser Service 的内容 copy 到 LunaSysMgr 进程中，下图是该过程示意图：

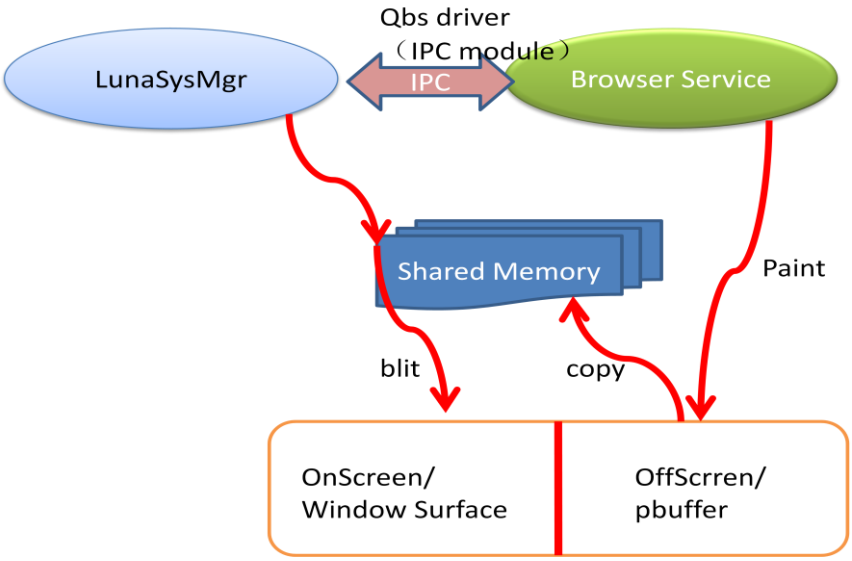


图 19　Browser App 绘制

大致过程是，Browser Service 进程 paint 内容到 pbuffer 中，并利用 IPC 机制将内容 copy 到 shared memory，LunaSysMgr 负责将 shared memory 中的内容 blit 到屏幕上。相对 Native App 来说不高效了，多了一次 copy 和 blit 的过程。

纵观 HP webOS 的 GUI 实现，其 Native App 设计是相当高效的；而所有 WebApp 由于运行在同一进程 LunaSysMgr 里，因此渲染也可以是很高效的；Browser App 稍微差点，但在大部分 GUI 系统里大多也需要经历这几步，因此也是情有可原的。然而，跟 Android 的 GUI 系统——SurfaceFlinger 相比，可以说 webOS 的做法毫无思想，杂乱无章，为了应用而生搬硬套了几类实现过程，现代化 GUI 系统中基于场景的渲染等技术也无体现，系统的可扩展性等也看不见端倪，Qt 的引入也感觉属于画蛇添足之举，况且 Qt 本身也是很重的 GUI。 随着应用的增多，HP webOS 是否能支撑呢？

6. Framework

Framework 可以说是整个 HP webOS 的重头戏。看了前面的内容，如果你举得 HP webOS 系统底层并不怎么样，那么现在可以告诉你，其 Framework 还是下了一番功夫的。

webOS 的 Framework 分为 SDK 和 PDK 两部分，SDK 用于 Web 开发，PDK 用于 Native 开发。

6.1. SDK

HP webOS 的 SDK 包括早期的 Mojo 和最新的 Enyo。

6.1.1. Mojo

6.1.1.1. Mojo package 格式

Mojo 应用程序是一个压缩包，展开以后形成一个子目录，子目录下的 appinfo.json 文件描述了有关 App 的信息，icon.png 文件代表了 App 的图标，index.html 是 App 的总体布局。还有 app、stylesheets、imagesd 等子目录分别存放与该 App 对应的代码和资源文件。

6.1.1.2. Mojo 框架

Mojo 应用程序包括三个组件，app、stage、scene，如下：

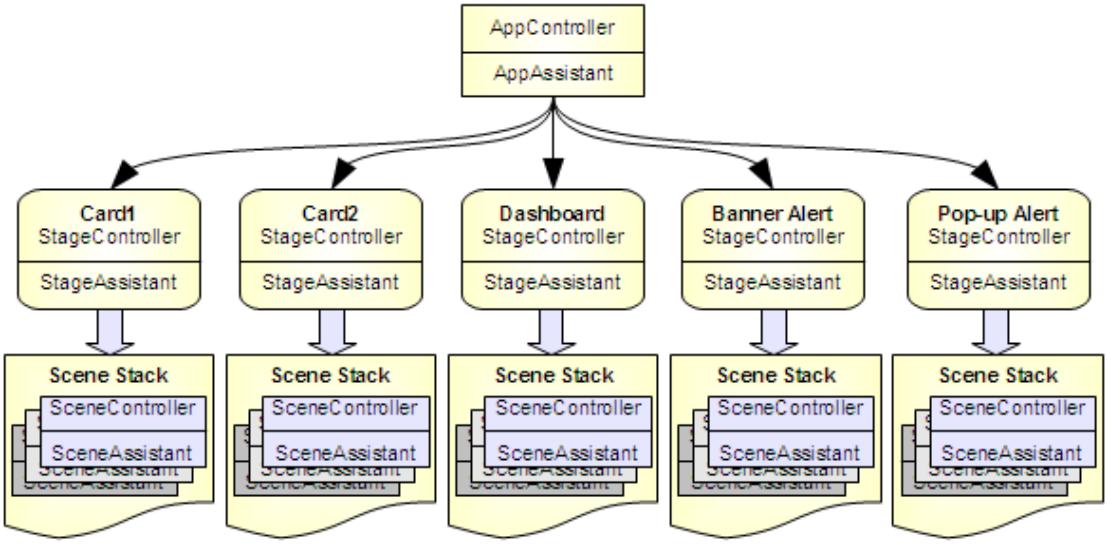


图 20 Mojo App 构成

- App：代表一个WebApp，每一个App至少包含一个Stage用来显示程序的主界面，可以包含多个Stage。
- Stage：是用来存储和展示Scenes的容器，代表了一个可以独立显示的窗体，相当于一个浏览器窗口或Tab窗口。该窗体中的内容由内嵌的scene来填充。Stage作为一个应用的基本显示单元，分为四种类型：Card、Dashboard、Banner Alert、Pop-up Alert。
- Scene：是用户可见界面，代表了一个与用户交互的Web页面（实质是个DIV）。Scene近似于一个传统的Web页面，有自己的HTML+CSS+JavaScript，显示在Stage中；Stage中可能存在多个Scene，每个Stage有一个Scene Stack结构，用来存储所包含的Scenes。Scene通过push和pop操作来完成切换：
  - Push(Activating)：将Scene放置到Stack的顶端显示出来的过程。

- Pop(Deactivating)：Stack上移除顶端Scene，当移除了顶层的Scene后，下面的Scene会显示出来。所有的Scene移除时，程序退出。

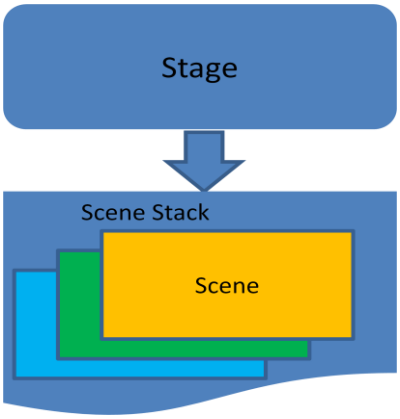


图21　Stage和Scene关系

Assistant 是一组 JavaScript 语言描述的属性和方法，提供给应用开发者调用 Controller 的功能的编程接口。Assistant 分为 AppAssistant、StageAssistant 和 SceneAssistant 三种类型。每一个 Controller 都有一个与之对应的 Assistant，应用开发者只能通过 Assistant 调用 Controller 提供的功能。Assistant 是用户对象，但有其特殊性，它是实现 Mojo 跟用户交互的核心对象。存在如下特点：

- Assistant class由用户（Mojo WebApp开发者）实现；
- Assistant不是对应UI对象的实例化，而是委托；实例化时，系统自动增加controller属性，指向对应的controller对象；通过它可访问controller对象。
- Assistant由Mojo实例化；
- Assistant应实现由Mojo所规范的接口。

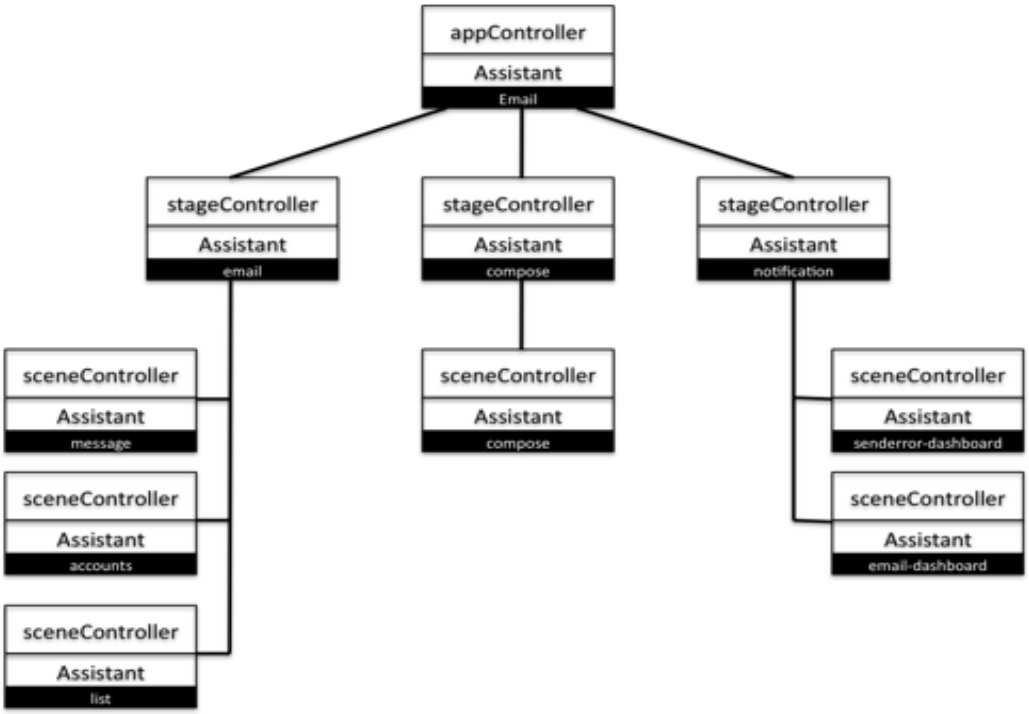


图 22 Controller 和 Assistants 关系图

6.1.1.3. Mojo 应用程序生命周期

WebApp 启动时首先创建一棵新的 DOM 树，并装载 index.html（或者 appinfo.json 的 main 属性中指定的网页），网页代码一般如下：

```
<script src="/usr/palm/frameworks/mojo/mojo.js"
  type="text/javascript" x-mojo-version="1" >
```

装载完毕将首先执行 mojo.js，它完成 prototype 库的初始化和相关 Framework 库的装载；接着装入 CSS 等其他资源文件。

Mojo Framework 将创建 application controller；如果 WebApp 提供了 application assistant，Mojo 还将实例化 application assistant 并调用其 setup 方法。

接着创建 stage controller，如果 WebApp 提供了 stage assistant，Mojo 还将实例化 stage assistant 并调用其 setup 方法。如果 WebApp 没有提供 stage assistant，或者其 stage assistant 没有 push 一个初始的 scene，则 Mojo Framework 将查找一个名叫 main 的 scene 并 push 到最顶端以供显示。

一旦一个 scene 被 push 完成，其生命周期将开始，完成如下动作：

- Mojo 查找相关 scene 的构造函数并调用此构造函数
- 增加 scene 到 DOM 树
- 如果存在对应 scene 的 assistant，调用其 initialize 方法；并给 Scene assistant 增加 controller 对象
- 获取 scene 的 view 模板并 load 到 DOM 中
- Mojo 允许用户切换 scene，这将产生 activate 和 deactivate 事件

下图是 scene 的生命周期：

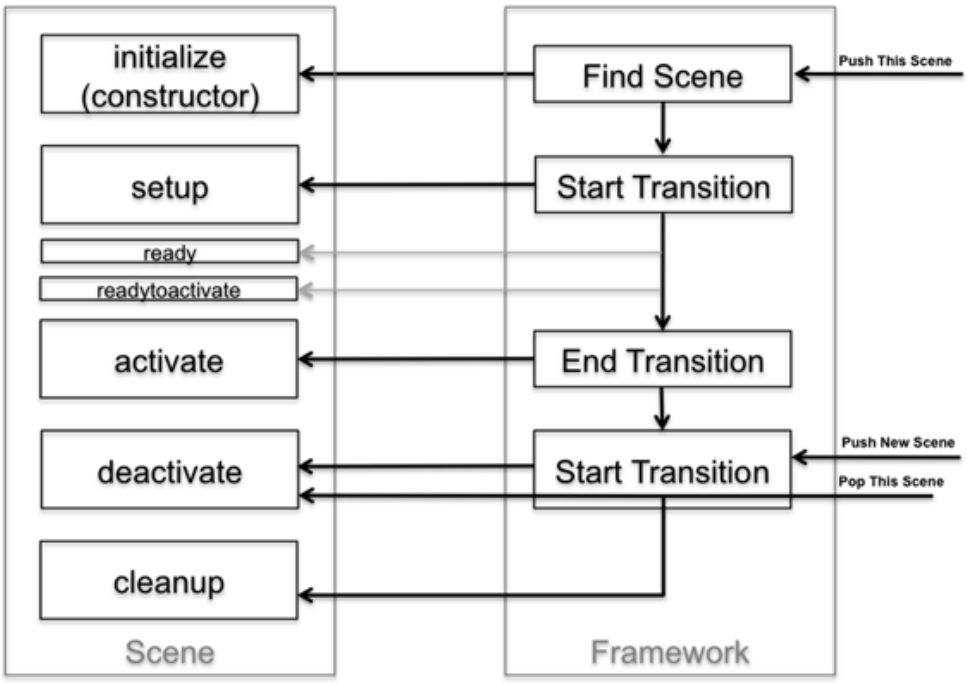


图 23 scene 的生命周期

6.1.2. Enyo

Enyo 是 webOS 3.0 发布的新的应用程序框架，这将代替原有的 Mojo。Enyo 是专为平板和“大屏手机”打造的。Enyo 除了针对更高分辨率与大尺寸显示器的设计外，还拥有硬件加速以及更快的启动速度（号称 1 秒内）。Enyo 的另一大亮点是自适应分辨率。Enyo 应用程序可以在一般计算机的浏览器里运行，不需要额外的仿真器；因此，开发人员可直接在浏览器中测试，甚至调试 Enyo 应用程序。从官方文档可知，相比 Mojo，Enyo 具备如下优点：

◆支持多种机型和不同屏幕尺寸；
◆加快程序启动速度；
◆面向对象，拥有更好的事件处理能力；
◆现代模块化设计——易于维护，代码重用；
◆与 Ares 配合良好（由同一个团队构建）；
◆可以基于浏览器开发，无需模拟器；
◆精简代码；
◆硬件加速。

迄今为止，Enyo 经历了 1.0 和 2.0 两个版本，Enyo 1.0 主要针对 HP TouchPad。关于 Enyo 的更多介绍请参见： http://enyojs.com/

6.2. PDK

HP webOS PDK 主要目标为：

- 方便移植 C/C++应用程序到 webOS 平台，尤其是使用 SDL 和 OpenGL ES 的游戏；
- 易于集成 C/C++ components，拓展 webOS 应用程序的能力。

PDK 程序需要在配置文件 appinfo.json 中指定其类型为 hybird 或者 pdk（即 Native 方式），如下：

```
{
  "type": "hybrid",
  "requiredMemory": <required memory in megabytes>
}
```

图 24 App type

Hybrid 模式是 PDK+Web 的混合开发模式，通过插件交互。其大致流程为：

- 定义插件标签

```
<object type="application/x-palm-remote" id="Plugin1" height=320 width=320
  x-palm-pass-event="true">
<param name="appid" value="com.palm.app.shapespin">
<param name="exe" value="shapespin_plugin">
<param name="Param1" value="0">
<param name="Param2" value="1">
</object>
```

- 实现插件特定处理函数

```
PDL_bool MyJSHandlerFunc(PDL_JSParameters *parms) { ... }
```

- 注册该处理函数

```
PDL_Err err = PDL_RegisterJSHandler("foo", MyJSHandlerFunc);
```

- 注册其他处理函数
- JavaScript 中调用插件处理中的函数

```
var element = document.getElementById("Plugin1");
element.foo();
// Note that you could also use this Mojo shortcut for the above two lines:
// $('Plugin1').foo();
```

- 返回值处理

```
PDL_bool MyJSHandlerFunc(PDL_JSParameters *parms) {
  int num =  PDL_GetNumJSParams(parms);
  if (num==2) {
      const char * firstName = PDL_GetJSParamString(parms, 0);
      const char * lastName = PDL_GetJSParamString(parms, 1);
      PDL_JSReply(parms, "Husband of Jane Doe");
      return PDL_TRUE;
  }
}
```

PDK 针对的开发方式是，使用 SDL 和 OpenGLES 提供的 API 进行开发，开发语言为 C/C++，支持显示处理、输入、多媒体、设备访问、系统服务、内存管理等。可以看得出来，HP webOS 在 Framework 层面投入了较多的精力，从这个角度看，HP webOS 应是“轻系统，重应用平台”。

7. 总结

由于当前 HP webOS 并未完全开源，不知道诸位 webOS 的研究者和爱好者在面对这样一个系统时会采取哪些方法一探究竟？会从哪些角度入手？ 这里介绍一下笔者所使用的方法，仅供参考。

- 代码阅读，尽管 HP webOS 尚未开源，但它使用了很多开源组件，其所有的开源组件及对应的 patch 是公开了的。本文中提到的很多实现细节就是根据其开源组件的 patch 并加以合理推测和适当试验加以验证的；
- HP webOS 开发文档；
- GDB/nm/反汇编的使用；
- 系统 Boot 系统分析，这里是 upstart；
- 使用 Strace 工具分析；
- 设计适当简单场景 demo 加以验证。

最后，从开发者角度看，抛却一切市场环境因素，究竟什么样的系统才是一个好系统，或者说至少应具备哪些特征？系统的上层（Framework）和底层又应如何取舍？留待各位思考。

8. 参考资料

[1] http://www.cocoachina.com/applenews/apple/2011/0819/3138.html，回顾 Palm webOS 平台发布历程。
[2] http://enyojs.com/
[3] http://opensource.palm.com
[4] http://isis-project.org/
[5] http://www.slideshare.net/unwiredben/javascript-on-hp-webOS-enyo-and-nodejs