tbrw2 应用内存优化方案

苏争光

2018.9.25

文档历史发放及记录

序号	变更（+/-说明）	作者	版本号	日期	批准
1	创建	苏争光	V0.1	2018/8/21	
2	补充 (AM 应用内存优化)	谢杰斌	V0.2	2018/8/27	
3	修改 (添加优化概览)	苏争光	V0.3	2018/8/29	
4	修改 (修改 AM 回收内存机制、应用权限设置)	张震	V0.4	2018/9/20	

目 录

1 引言 **4**

1.1 文档目标 4

1.2 预期读者 4

1.3 优化措施总览 4

2 TBrowser2.0 内存优化 **5**

2.1 优化方法 5

2.2 实现原理 6

2.2.1 MemoryPressureListener 6

2.2.2 MemoryPressureMonitor 7

2.3 AM 判断应用优先级 8

3 AM 应用内存优化 **9**

3.1 AM 循环检测系统内存 9

3.2 AM kill 后台应用流程 10

4 非 AM 应用管理与内存优化 **11**

4.1 非 AM 应用管理 11

4.2 NotAMApp 开机启动流程 11

4.3 NotAMApp 退出重启流程 12

4.4 部分实现代码 12

5 tbrw2 应用优化 **14**

5.1 输入法方案优化 14

5.2 Netflix 监听 AM 内存广播 14

5.3 提供设置 ipad useragent 的页面 14

5.4 巧用 TbrowserActivity 基类 15

5.5 hbbtv render 优化 15

1 引言

1.1 文档目标

目前海外电视 UI 和各类 app 都是基于 HTML5 开发的网页应用，然后由 TCL 自主研发的 TBrowser2.0 打开这些页面。这些三方页面一般都是为 PC 设计的，对内存要求较多，而我们海外平台系统普遍内存较少。以 563 平台为例：

563 平台总物理内存 768M，硬件占用：305M，软件可用：463M。其中 UI 占用 50M 内存 sitatvservice，hbbtv，tplayer，dial，appmanager 等其他应用也会占用不少内存，在启动 TBrowser 之前系统可用内存仅为 260M。

因此本文的目标是指导 tbrw2 应用开发人员使用下面介绍的几种内存优化方法，以提高系统整体的稳定性和可用性。

1.2 预期读者

tbrw2 应用开发人员

1.3 优化措施总览

序号	优化措施	测试平台	优化内存(单位M)	测试方法
1	添加YouTubeLauncher.ini	MS6586	20	播放4K片源30分钟查看render_youtube进程占用RSS
2	AM内存不足kill后台AM应用	MS6586	100+	进入TBrowser，打开MSN页面浏览，消耗系统内存。 AM会kill后台AM应用，如livetv、launcher。
3	Netflix监听AM内存不足广播自动退出	MS6586	80	进入TBrowser，打开MSN页面浏览，消耗系统内存。 AM会发系统内存不足的广播,Netflix后台应用会退出释放内存
4	非AM应用监听AM内存不足广播自动退出	NT563	30	进入TBrowser，打开MSN页面浏览，消耗系统内存。 AM会发系统内存不足的广播,非AM应用会退出释放内存。
5	输入法方案优化	MS6586	4	打开Tbrowser应用，查看地址栏对应render的RSS。 对比注入实现的输入法和现在输入法的实现
6	去掉hbbtv render	MS6586	30	开机hbbtv render默认不打开

图 1.1: 优化措施总览

2 TBrowser2.0 内存优化

2.1 优化方法

```
# pwd
/system/tbrowser2/config
# ls
CommonWebApp.ini                default.ini
HTMLParserUrlList.ini           hbbtv_tbrowser.ini
WebBrowser.ini                  hbbtv_tbrowser_error_code.ini
WebLauncher.ini                 inject
YouTubeLauncher.ini             kioslaverc
YoutubeKeyMap.ini               systemUi.ini
addressInput.ini
# cat systemUi.ini
video_max_width=3840
video_max_height=2160
allow-file-access-from-http=0
enable-download-mode=0
moderate_pressure_memory_use=30
critical_pressure_memory_use=45
set_call_as_function_handler=1
#
```

图 2.1: 6586 配置文件

如图所示，每个 render 在 tbrowser2/config 目录下都应该有对应的 page_name.ini，render_name 与调用 tos_tbrowser 接口所传的 page_name 一样。在这个 ini 应该要包含 moderate_pressure_memory_use 和 critical_pressure_memory_use 两个配置项。

其中 moderate_pressure_memory_use 代表 render 内存使用超过这个值浏览器将开始回收内存，critical_pressure_memory_use 代表 render 内存使用超过这个值浏览器将尽量多的回收内存。

这两个配置应该根据 render 的不同设置合理的值，比如 render_systemui 内存使用在 35M-50M 之间，那么这个值应该设置为 40M-45M。而 render_hbbtv 内存使用普遍在 120M-220M，那么这个值应该设置为 160M-200M。这样既可以限制 render 的峰值内存使用，同时又避免浏览器频繁回收内存造成用户体验不好。

优化效果：systemui 添加收阈值 30-45，原来没有设置 render_systemui 峰值 55M，添加阈值之后峰值 41M。预计节约 14M。

2.2 实现原理

2.2.1 MemoryPressureListener

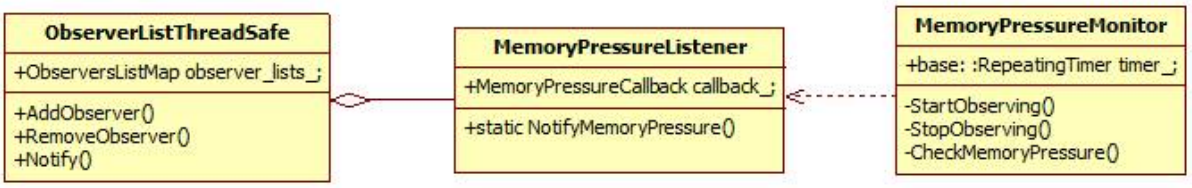

图 2.2: 浏览器 memory_listener

1、浏览器内部模块如果需要进行内存回收，就在类中添加 MemoryPressureListener 成员变量，并设置 OnMemoryPressure 回调函数

2、MemoryPressureListener 在构造函数调用 g_observers.Get().AddObserver(this);

3、MemoryPressureMonitor 检测内存，通过 MemoryPressureListener 发送消息给所有 MemoryPressureListener 的子类

2.2.2 MemoryPressureMonitor

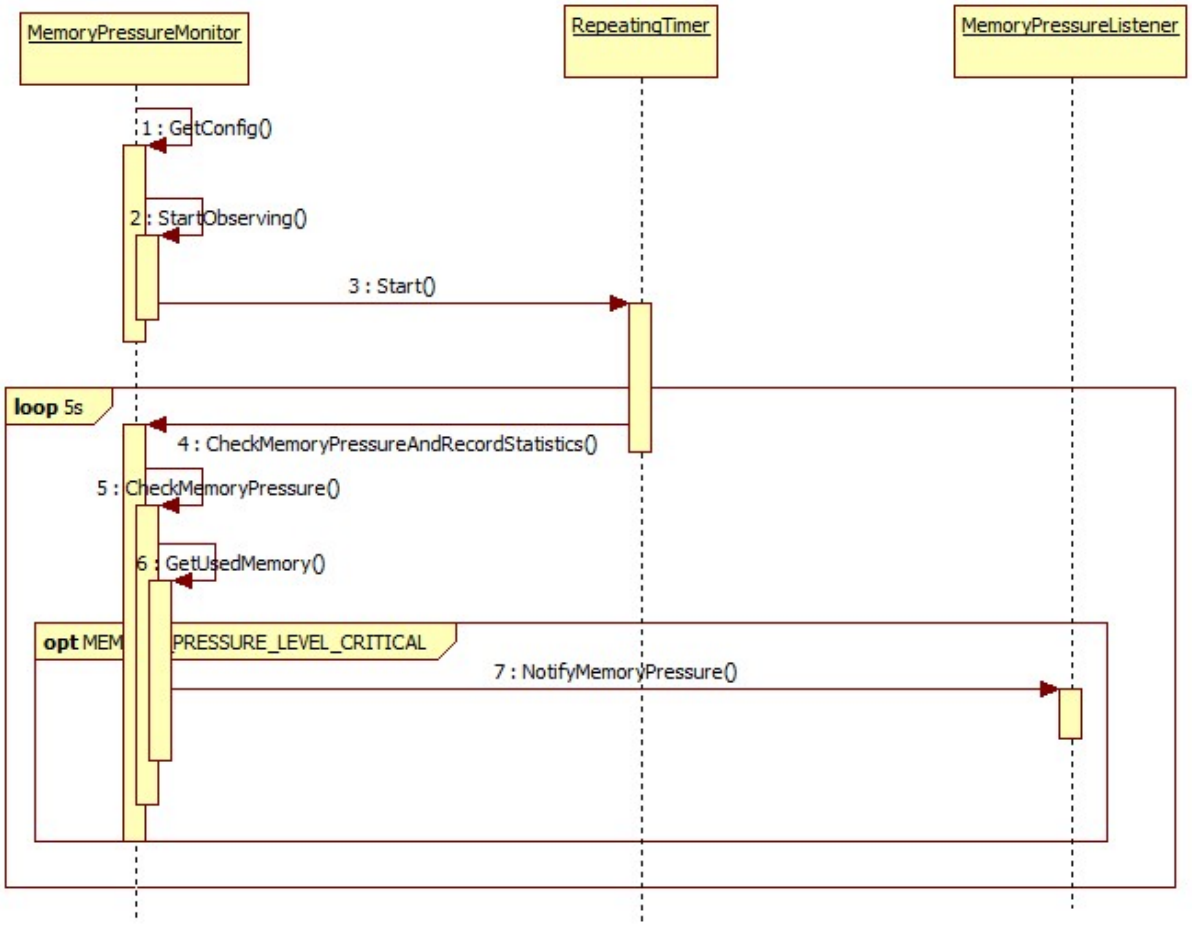

图 2.3: 浏览器 memory_monitor

1、getconfig 获取 ini 文件里面配置的 render 回收的阈值 moderate_pressure_memory_use 和 critical_pressure_memory_use

2、MemoryPressureMonitor 初始化时启动一个 timer，循环调用 CheckMemoryPressure-AndRecordStatistics 用来判断内存使用情况

3、GetUsedMemory 通过获取系统信息得到浏览器进程使用内存，通过与回收阈值对比得到是否应该进行内存回收

4、当内存使用超过 moderate_pressure_memory_use 发送 MEMORY_PRESSURE_LEVEL_MODERATE 给所有注册的 listener，此时 listenser 回收可回收内存的一半

当内存使用超过 critical_pressure_memory_use 发送 MEMORY_PRESSURE_LEVEL_CRITICAL 给所有注册的 listener，此时 listenser 回收可回收内存的一半

2.3 AM 判断应用优先级

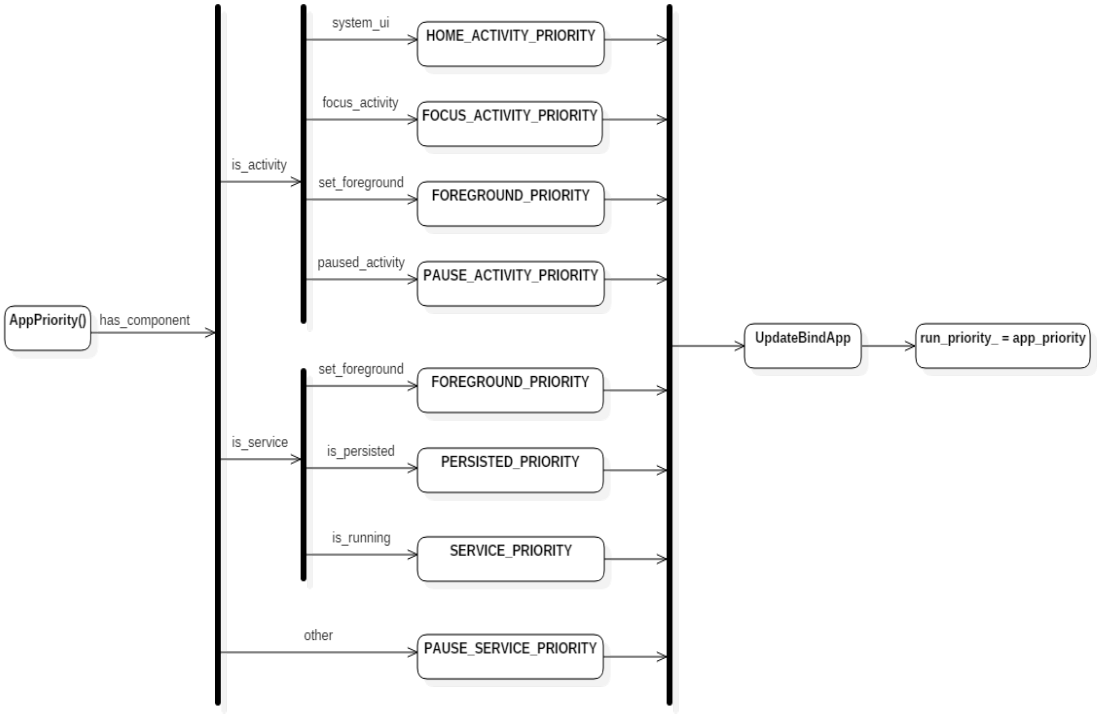

图 2.4: AM 判断应用优先级

AM 会逐个判断应用是否是 systemui、是否调用了 tos_am_setForeground 接口，如果是这样的应用优先级较高，则不能被 kill，否则会被 kill。

将可以被 kill 的应用按照优先级划分成三组，按照系统内存状态以组为单位关闭应用。

1. 第一组：paused 的 activity，running 的 service。
2. 第二组：persisted 设置为 true 的 service。
3. 第三组：focus app。

3 AM 应用内存优化

3.1 AM 循环检测系统内存

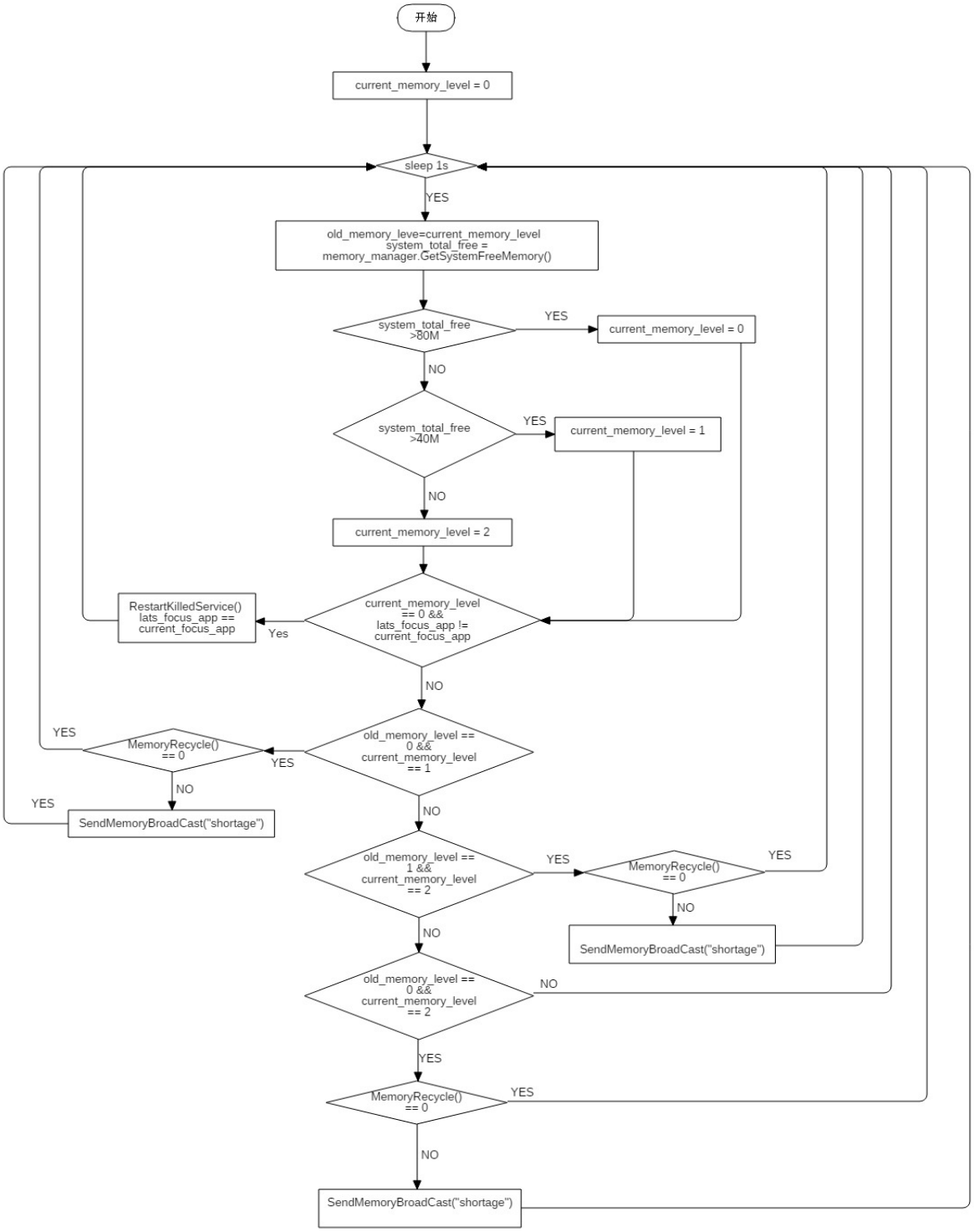

图 3.1: AM 循环检测系统内存

AM 启动参数中设置 recycle_size 和 recycle_critical_size，不同平台可以设置不同的值

- 系统 free memory 大于 recycle_size，依据系统剩余内存大小，设置内存状态等级为 0，可以尝试启动不是在当前应用活动状态下退出的应用进程。
- 系统 free memory 大于 recycle_critical_size，小于 recycle_size，设置内存状态等级为 1。
- 系统 free memory 小于 recycle_critical_size，设置内存等级为 2。

3.2 AM kill 后台应用流程

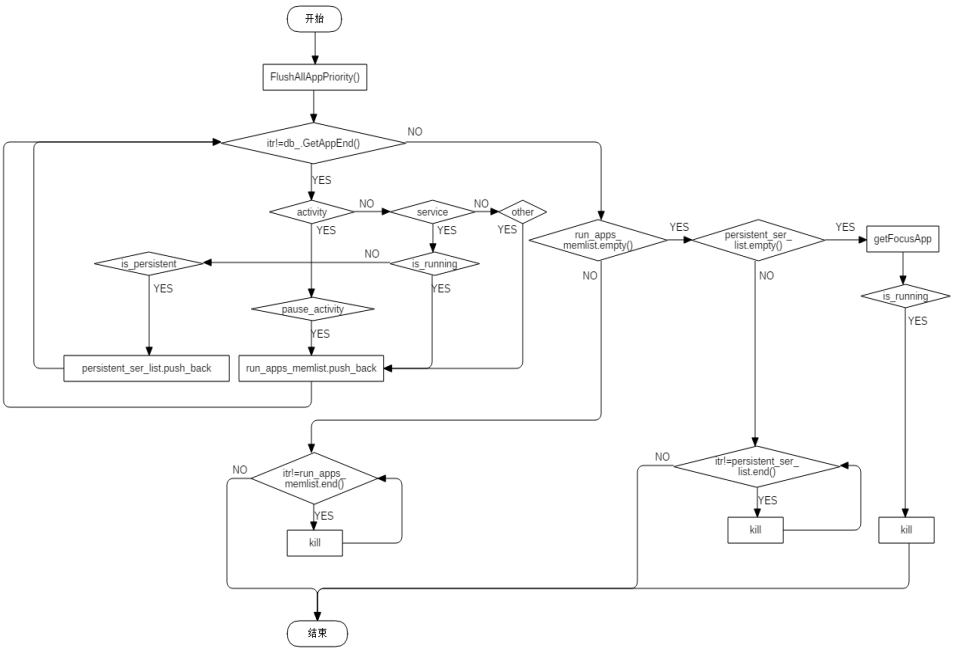

图 3.2: AM kill 后台应用流程

依据系统内存状态，选择退出不同优先级的 activity 或者 service。

当状态由 0 变为 1 时，kill 掉 paused 的 activity，running 的 service，并且发送系统内存广播，传递系统剩余内存大小信息。

当状态由 1 变为 2 时，kill 掉 persisted 设置为 true 的 service。并且发送系统内存广播，传递系统剩余内存大小信息。若此后状态仍然为 2，那么为了保证系统稳定，需要将当前 focus 的 app 杀掉。

当状态由 0 变为 2 时，先 kill 掉第一组应用，发送系统内存广播，传递系统剩余内存大小信息。若内存状态变为了 1，那么就暂时不做处理，若状态仍持续为 2，则 kill 掉 persisted 设置为 true 的 service。

4 非 AM 应用管理与内存优化

4.1 非 AM 应用管理

非 AM 应用由一个开机启动的 service 管理，这个 service 为普通服务，管理若干非 AM 应用，无法退出的非 AM 应用无法被该 service 管理。

4.2 NotAMApp 开机启动流程

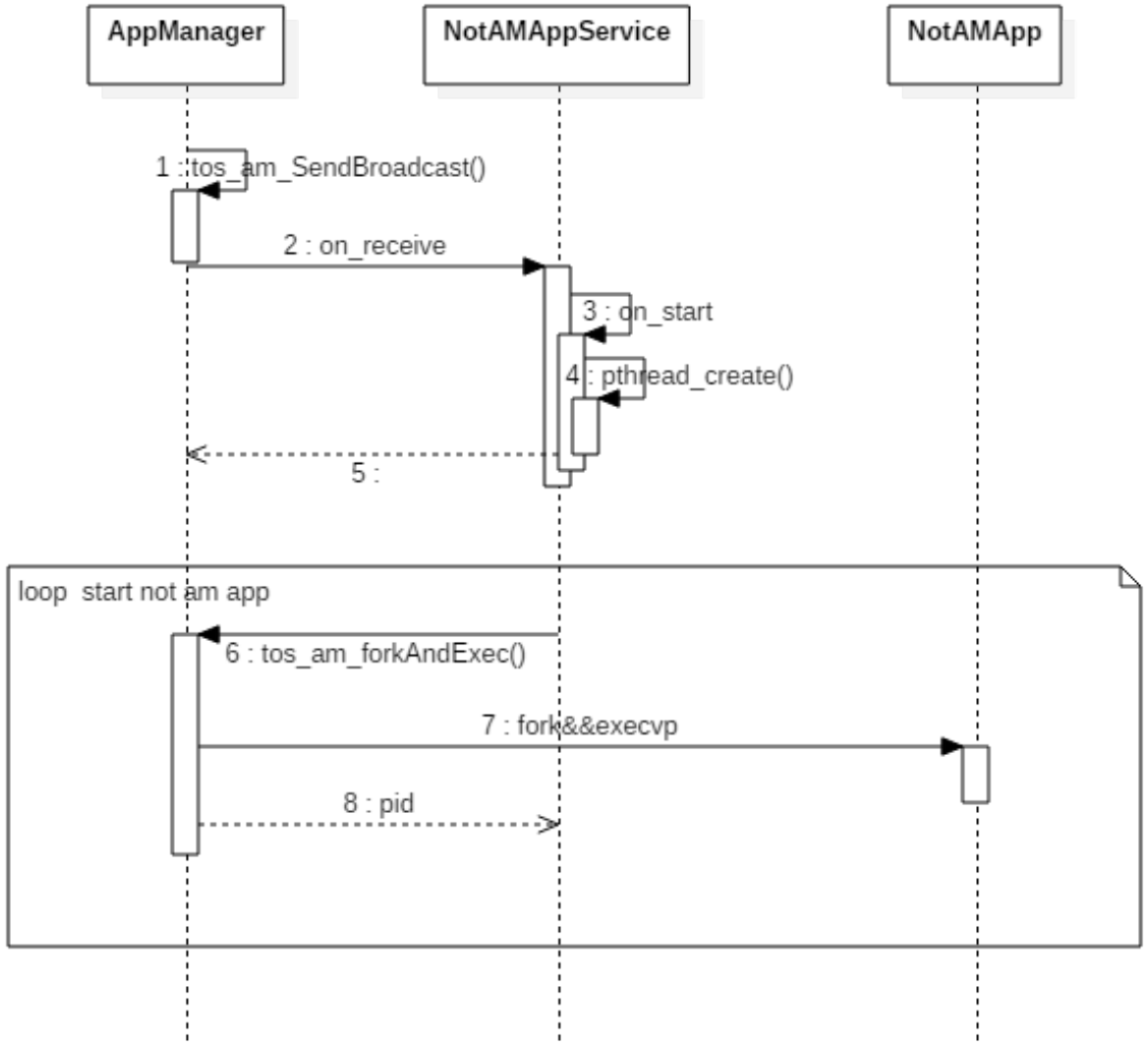

图 4.1: NotAMApp 开机启动时序图

NotAMApp 服务启动后创建一个线程去启动管理的应用，避免拖慢开机速度。

4.3 NotAMApp 退出重启流程

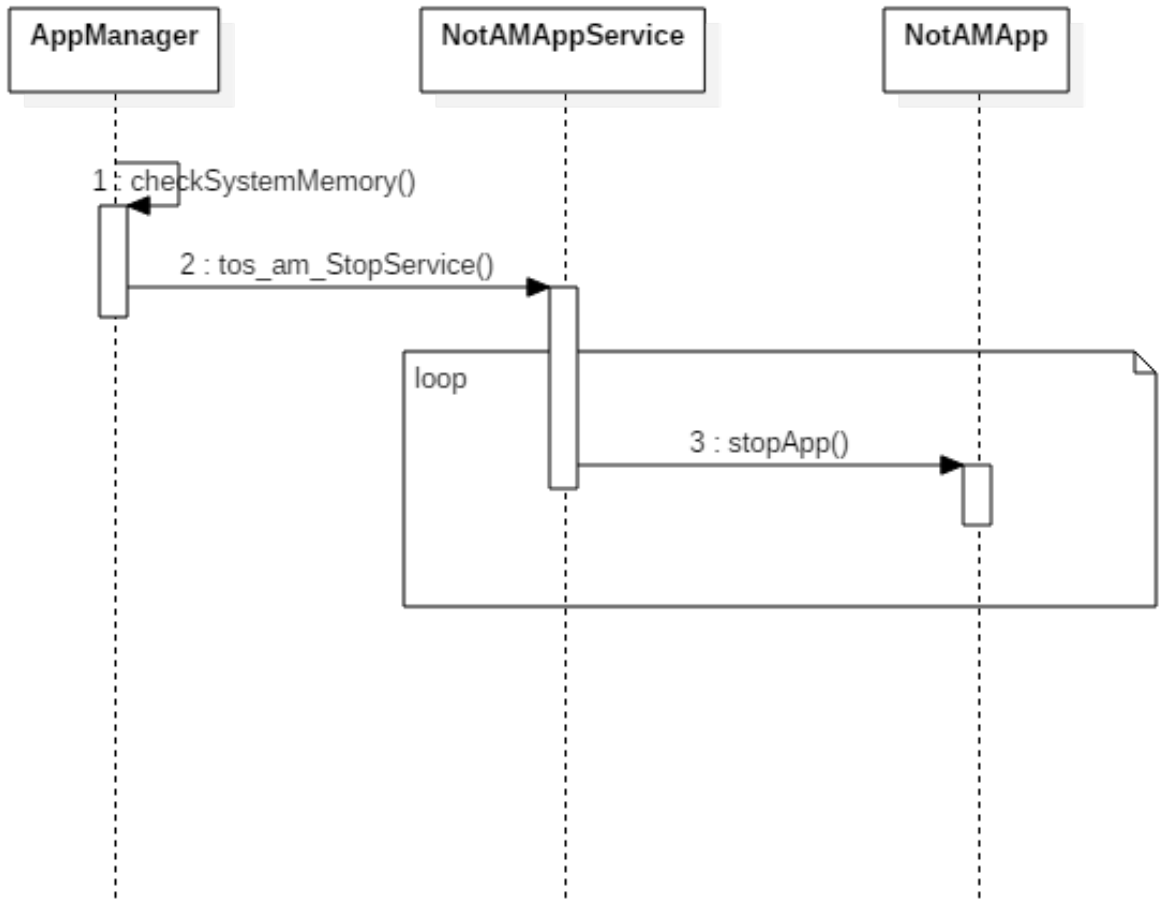

图 4.2: NotAMApp 退出重启时序图

NotAMApp 退出时，会将管理的所有非 AM 应用关闭。

4.4 部分实现代码

```
1
2 int MemoryMonitorBroadcastOnReceive(tos_am_Contex ctx, void *ud) {
3   tos_am_Intent intent = {0};
4   intent.size = sizeof(tos_am_Intent);
```

```
  tos_am_getIntent(ctx, &intent);
  int memory_level = atoi((char*)intent.data);
  char focus_app[128] = {0};
  tos_am_getFocusApp(ctx, focus_app, 128);
  if (memory_level == TOS_AM_MEMORY_PRESSURE_LEVEL_NONE){//内存充足，
   准备重启应用
    startApp(focus_app);
  } else if(memory_level == TOS_AM_MEMORY_PRESSURE_LEVEL_MODERATE){
   //内存紧张，开始kill优先级低的应用
    stopApp(focus_app,Priority_Low);
  } else{//内存严重不足，开始kill所有应用
    stopApp(focus_app,Priority_High);
  }
}

//添加监听内存变化的广播
void init(){
  tos_am_Intent broadcast_intent = {0};
  memory_shortage_broadcast_.on_receive =
   MemoryMonitorBroadcastOnReceive;
  snprintf((char*)broadcast_intent.action,
   tos_am_Intent_ACTION_MAX_LENGTH, "tvos.broadcast.memory.shortage"
   );
  tos_am_addBroadcast(contex_, &broadcast_intent, (void*)&
   memory_shortage_broadcast_);
}

//打开应用函数
void startApp(const char* focus_app) {
  for(int i = 0; i < app_vector_.size(); i++) {
  //如果前台应用没有改变，就不重启应用。避免频繁启动退出应用
    if (app_vector_[i].pid == 0 && app_vector_[i].focus_app !=
   focus_app) {
      std::string app_path_name = app_vector_[i].app_path +
   app_vector_[i].app_name;
      char* const args[] = {(char*)app_vector_[i].app_name.c_str(),
   NULL};
      app_vector_[i].pid = tos_am_forkAndExec(app_path_name.c_str(),
   args);
    }
  }
}
```

```
//关闭应用函数
void stopApp(const char* focus_app,Priority_Type priority){
  for(int i = 0; i < app_vector_.size(); i++) {
    if (app_vector_[i].pid && app_vector_[i].priority == priority) {
      char command[32] = {0};
      snprintf(command, sizeof(command) - 1, "kill -9 %d",
    app_vector_[i].pid);
      os_cmd_system(command);
      app_vector_[i].pid = 0;
      app_vector_[i].focus_app = focus_app;//记录关闭应用时focus app
    }
  }
}
```

5 tbrw2 应用优化

5.1 输入法方案优化

原输入法是以注入的形式注入到页面中使用，并且在每一个用到输入法的应用中均需要包含对输入法处理的代码，会造成代码冗余。新输入法代码只包含一份，便于管理，并且可以节约 4M 左右的内存。

新输入法只适用于基于 TbrowserActivity 类开发的应用。

5.2 Netflix 监听 AM 内存广播

Netflix 常驻后台会占用 80M 左右内存，通过监听 tvos.broadcast.memory.shortage 广播，在内存使用级别为 1 时主动退出，能释放 80M 的内存。

5.3 提供设置 ipad useragent 的页面

TBrowser 应用提供了设置 UA 的选择框，可以选择 pc UA 或者 ipad UA。使用 ipad UA 访问某些页面可以节约内存。

5.4 巧用 TbrowserActivity 基类

TbrowserActivity 基类实现了中间件消息回调函数、浏览器消息回调函数、输入法等基础功能，子类如果使用了这些方法能少写很多代码。

5.5 hbbtv render 优化

原来 hbbtv 开机启动会调用 tos_tbrowser_load_url_with_name(PAGE_NAME, about:blank); 导致后台一直有 hbbtv 的 render 进程。

现在代码已经去掉这句话，等真正有需要打开 hbbtv 页面的时候直接打开正确的页面，这样可以节约 30M 左右内存。