android_system_properties

Last edited by **caoquanli** 1 month ago

Android 8.1 property系统属性

Table of Contents

1. 概述

1.1. 安卓属性机制

1.1.1. 属性原理

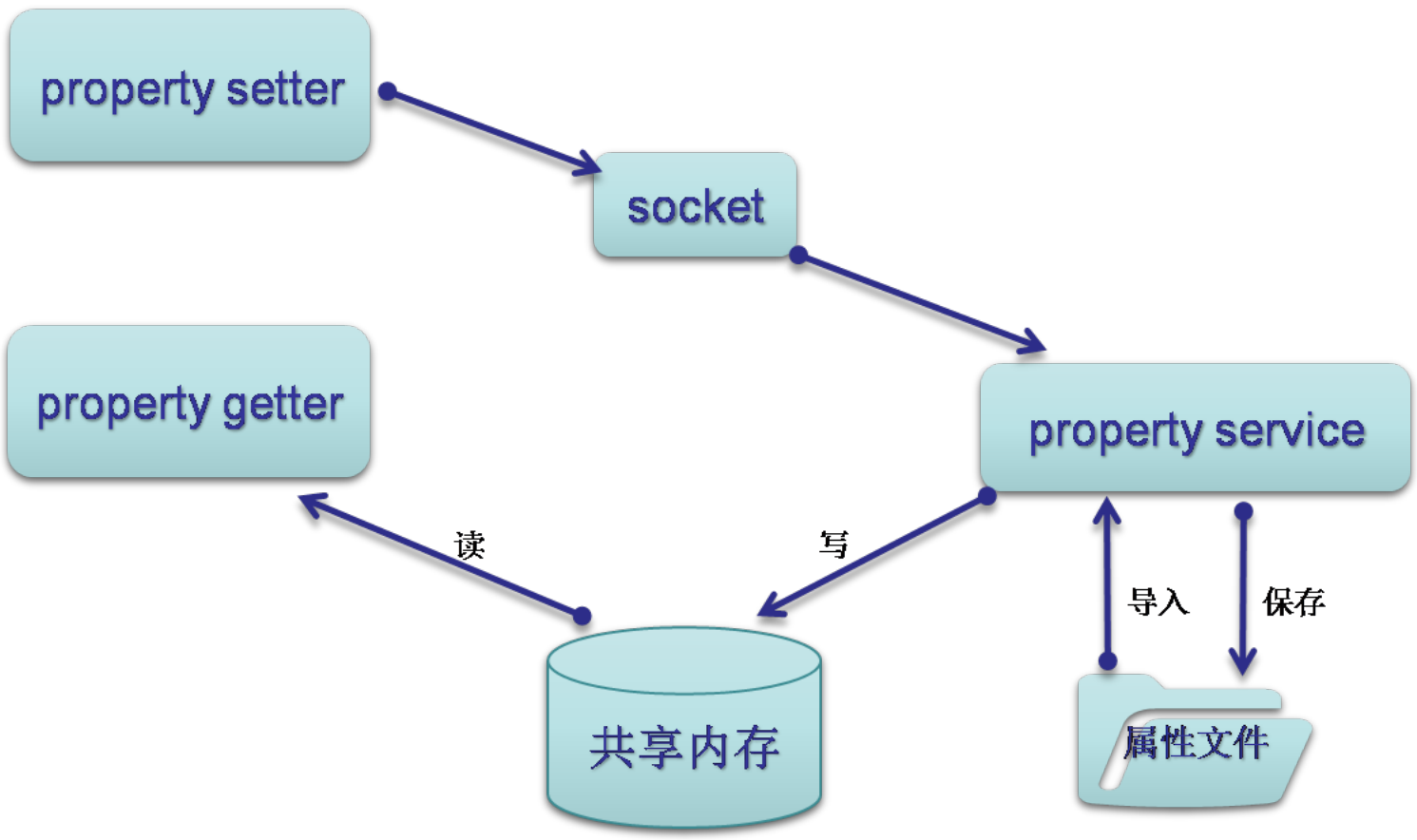
- Android系统属性是在系统运行过程中用于进程间共享的各种设置值。可以简单地看作为运行在进程间的全局变量，这个变量对所有进程可见，但只能由init进程添加和修改，其他进程可以通过socket向init进程发出添加或者修改的请求。
- 每个属性都有一个key和value，以"键=值"形式保存在init进程创建的共享内存当中，并由init进程的子线程property server进行管理。其他进程在启动的时候将该共享内存映射到自己进程的虚拟地址空间，通过封装的共享内存区的接口实现指定属性值的获取。

1.1.2. 属性作用

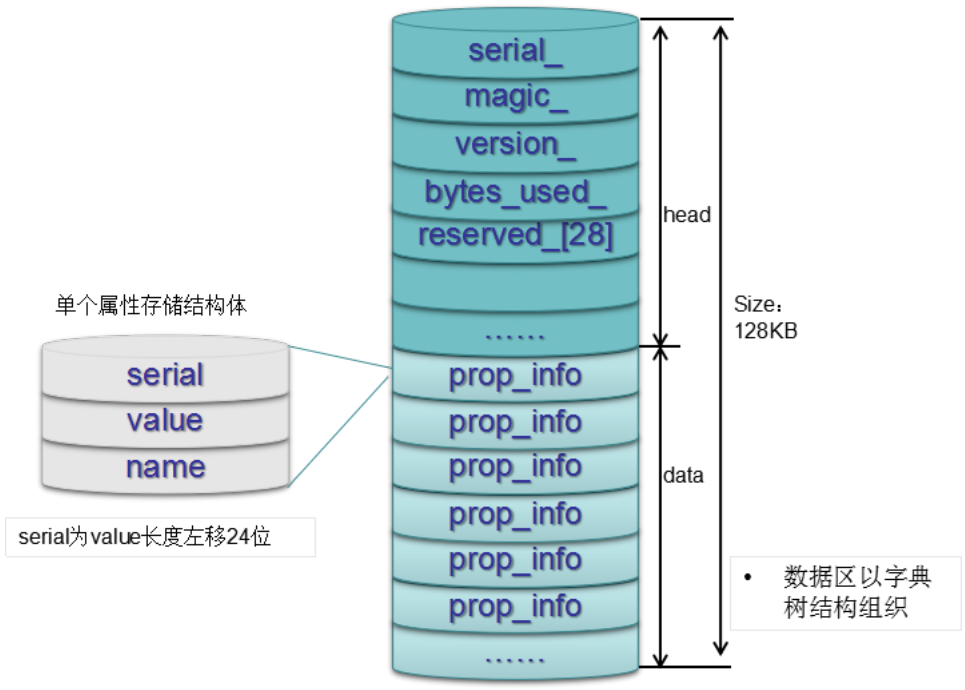
- 用于进程间共享设置值：当一个进程设置完属性值后，另一个进程可以随时读取该属性值做出相应动作。
- 控制服务（native进程）的启动与停止：直接通过set接口发送ctl.xx控制指令，实现对init进程启动的服务的控制。
- 触发init进程的action：定义在.rc文件中的属性action，可以在属性设置的时候通知init进程执行action对应的动作。
- 控制机器的关机、重启：专用的"sys.powerctl"属性，可以通过代码或者终端发出关机、重启请求。
- 永久保存各种设置值：persist属性设置时可以保存到文件，实现重启后任然有效。

1.1.3. 属性框架

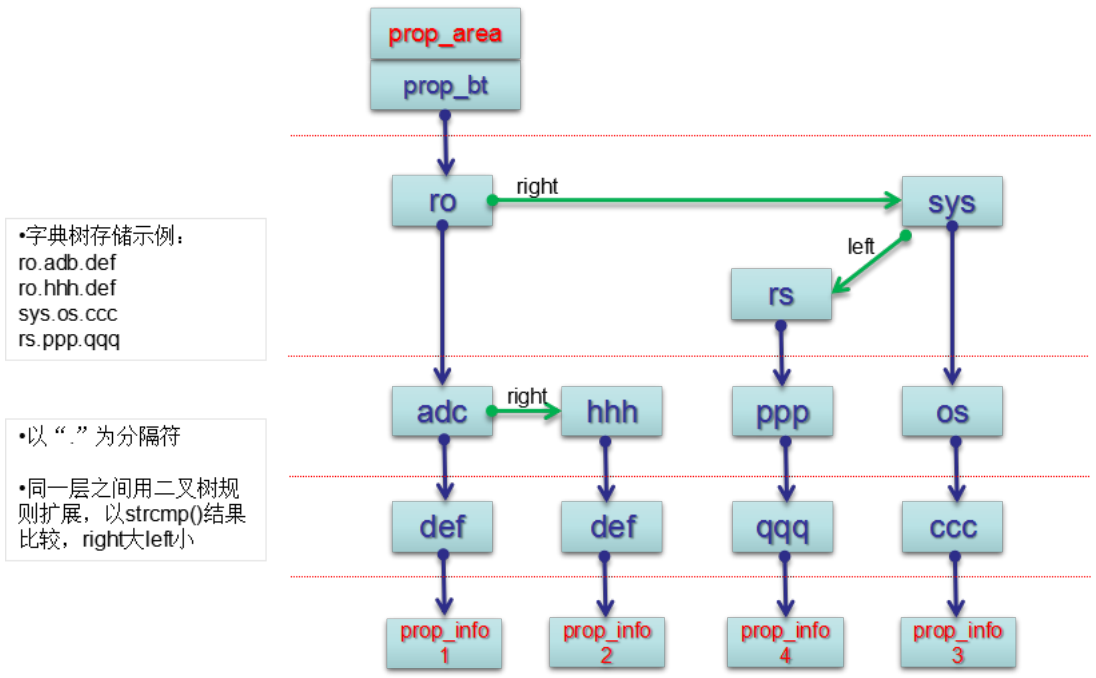
- Android属性的框架也可以简单地看成生产者和消费者的关系表达，生产者（调用set的进程）通过管理者（property service）向商店（共享内存）提供公开产品（只读属性），消费者（调用get的进程）直接到商店去消费（不能拿走）。此外，管理者还会可以从仓库中（属性文件）中拿取或者保存产品。
- 一个常见的Android属性框架图如下所示：



- 在这份框架图中，共享内存时init进程创建的，而property service也是init进程的一个子线程。共享内存的大小限定为128KB，其结构包括一小段记录共享内存区状态的head和一大段数据区，每个属性都以一个prop_info的结构体封装。基本结构如下：



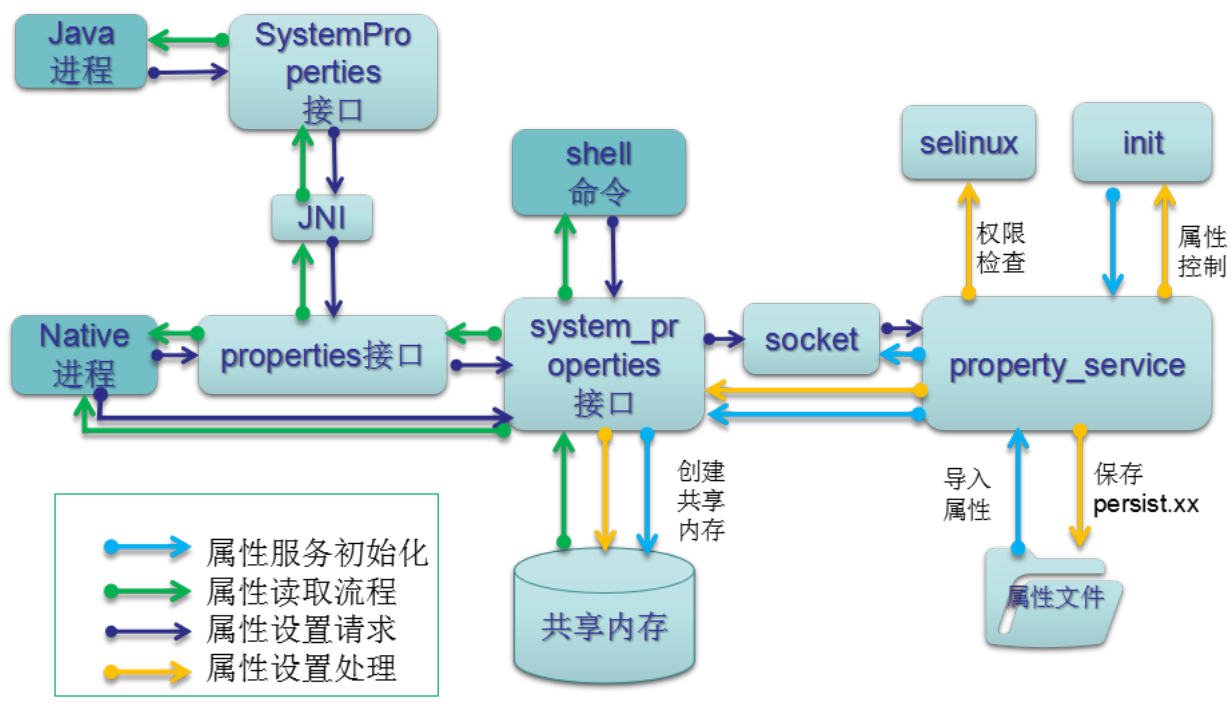
- 数据区是以字典树的形式组织的，查找和插入都遵从字典树的结构，这里找了一个网上的示例如下：



- 图中，每一个属性信息都相当于一片叶子，属性名以"."分割成为一层层枝点，每片叶子都可以通过属性名一层层查找得到。对应同一层的枝点按照二叉树的规制扩展，通过strcmp()比较ascii值，数值大的扩展为右分支，数值小的扩展为左分志。
- 参考资料：
  - 深入讲解Android Property机制
  - android SystemProperty系统属性分析

1.2. 属性的使用

- Android属性系统由init进程管理，提供了属性设置和控制接口，接口使用前的操作包括属性服务初始化、共享内存的创建、属性文件的加载、客户端映射共享内存等。属性的使用则主要是设置和获取接口的调用过程。在属性框架图上，可以扩展出一张各模块的关系图如下：



- 图中客户端包括了java进程、native进程与shell端工具（setprop，getprop）。system_properties提供了Android系统属性的综合管理接口，包括属性的读取、设置、共享内存的创建、初始化等等，可以提供客户端和属性服务端共同使用。
- 在属性服务启动、共享内存创建后，客户端进程在加载libc.so的时候会通过system_properties接口将共享内存映射到本进程的虚拟地址空间，之后进程运行过程中就可以通过各级属性接口最终访问到该共享内存。
- 属性的使用也就是三个客户端对各自接口的调用。

1.2.1. Java层接口

- java层属性设置接口封装在android.os.SystemProperties 包中，主要的设置接口如下表：

接口	注释
String get(String key)	获取属性，如果key为null，则返回null；如果key不存在，返回空字符串""

String get(String key, String def)	获取属性，如果为空，返回默认值
int getInt(String key, int def)	获取属性，返回整数值
long getLong(String key, long def)	获取属性，返回长整型
boolean getBoolean(String key, boolean def)	获取属性，转换为bool型。返回true的情况："1"/"y"/"yes"/"true"/"on" 返回false的情况："0"/"n"/"no"/"false"/"off"
void set(String key, String val)	设置属性，key长度小于32，val小于92
void addChangeCallback(Runnable callback)	注册属性变化回调函数（依赖于下面的属性报告
void reportSyspropChanged()	报告属性变化（调用回调）

- *后面两个接口虽然也是封装在一个包内，但是和Android属性系统应该没有直接关系。

1.2.2. Native层接口

- native层的接口主要封装在properties中，调用时包个头文件#include "cutils/properties.h"，主要接口如下：

接口	注释
int property_get(const char *key, char *value, const char *default_value)	获取属性，得到属性长度小于等于0 ，则返回默认值
int property_set(const char *key, const char *value)	设置属性，返回0成功，返回-1失败
int32_t property_get_int32(const char *key, int32_t default_value)	获取属性，转换为指定格式，转换失败返回默认值，超出界限返回边界值
int64_t property_get_int64(const char *key, int64_t default_value)	获取属性，转换为指定格式，转换失败返回默认值，超出界限返回边界值
int8_t property_get_bool(const char *key, int8_t default_value);	获取属性，转换为0/1，返回true(1)的情况："1"/"y"/"yes"/"true"/"on" 返回false(0)的情况："0"/"n"/"no"/"false"/"off"

- 上面的接口文件都基于system_properties接口进行了一定的封装，native进程也可以直接调system_properties接口设置和查找属性，使用时需添加#include <sys/system_properties.h>头文件，主要接口如下：

接口	注释
int __system_property_set(const char* key, const char* value)	设置属性，属性存在时更新，不存在则创建；返回0成功，返回-1失败
int __system_property_get(const char* name, char* value)	获取属性，返回属性值长度
prop_info* __system_property_find(const char* name)	查找共享内存中的属性节点
int __system_property_read(const prop_info* pi, char* name, char* value)	从节点中读出属性值
bool __system_property_wait(const prop_info* pi, uint32_t old_serial, uint32_t* new_serial_ptr, const struct timespec* relative_timeout)	挂起等待属性值变化，超时relative_timeout返回false

1.2.3. shell工具

- shell终端可以输入getprop、setprop指令设置或者查找属性。

指令	注释
getprop	列出所有属性
getprop key	获取指定key的值
setprop key value	设置一个属性，如果属性不存在则创建

1.2.4. 其他

- 对应属性名称可以包含的特殊符号只有"." "-" "@" ":" "_"
- 当前系统版本的属性名称长度限制在32字节以下，属性值长度为92字节以下，共享内存128KB
- 如果希望在编译阶段把属性编进属性文件，只需在相应的mk文件中加入类似语句：

```
PRODUCT_PROPERTY_OVERRIDES  += ro.custormize.version=123456
```

- 具体的属性服务启动、属性设置、属性读取流程下节继续展开。

2. 属性服务的启动

2.1. init进程初始化属性服务

- init进程在启动的时候就在其main()中开始做一系列Android系统关键的初始化工作，包括创建和挂载设备节点、初始化selinux、启动内核log系统等，之后调用./system/core/init/property_service.cpp文件中的property_init()接口属性初始化。
- property_init()直接调用到./bionic/libc/bionic/system_properties.cpp文件中的__system_property_area_init()。

2.2. 创建属性共享区

- *system_property_area_init()函数首先创建了"/dev/*properties__"共享文件夹，之后在map_prop_area_rw()函数中打开刚刚创建的共享文件夹，拿到文件描述符。
- 调用mmap()将文件描述符传递进去，以可读可写的方式将该共享文件夹映射为共享内存。

2.3. 加载默认属性文件

- 共享内存创建后，先通过property_service.cpp文件中的property_set()创建一些与boot以及内核相关的不可修改属性ro.xx，加载到共享内存中。
- 之后调用property_load_boot_defaults()接口导入默认属性文件，默认属性文件目录包括：

```
"/system/etc/prop.default"
"/prop.default"
"/default.prop"
"/odm/default.prop"
"/vendor/default.prop"
```

- 属性文件的导入操作就是将一些在编译阶段就写进属性文件的属性读取并加载到共享内存当中。加载过程也是调用的property_set()接口。

2.4. 启动属性设置的socket

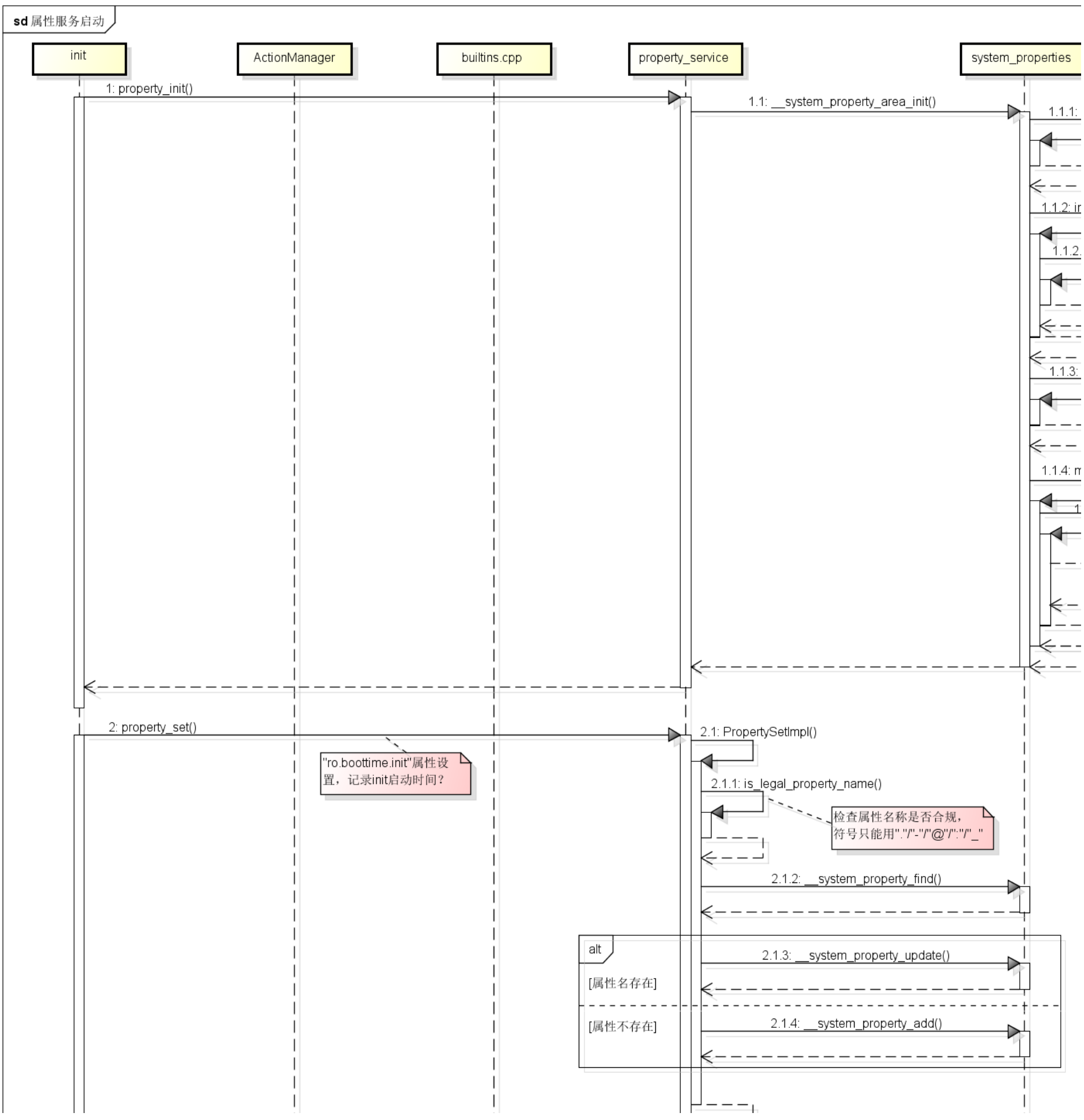
- 默认属性文件加载完毕之后，init进程调用start_property_service()来启动服务。
- start_property_service()函数一开始调用了property_set()创建"ro.property_service.version"属性标记属性服务的版本，这个版本号涉及到客户端与属性服务的通信协议。
- 之后通过CreateSocket()创建"/dev/socket/property_service" 套接字，循环监听客户端属性设置的请求。

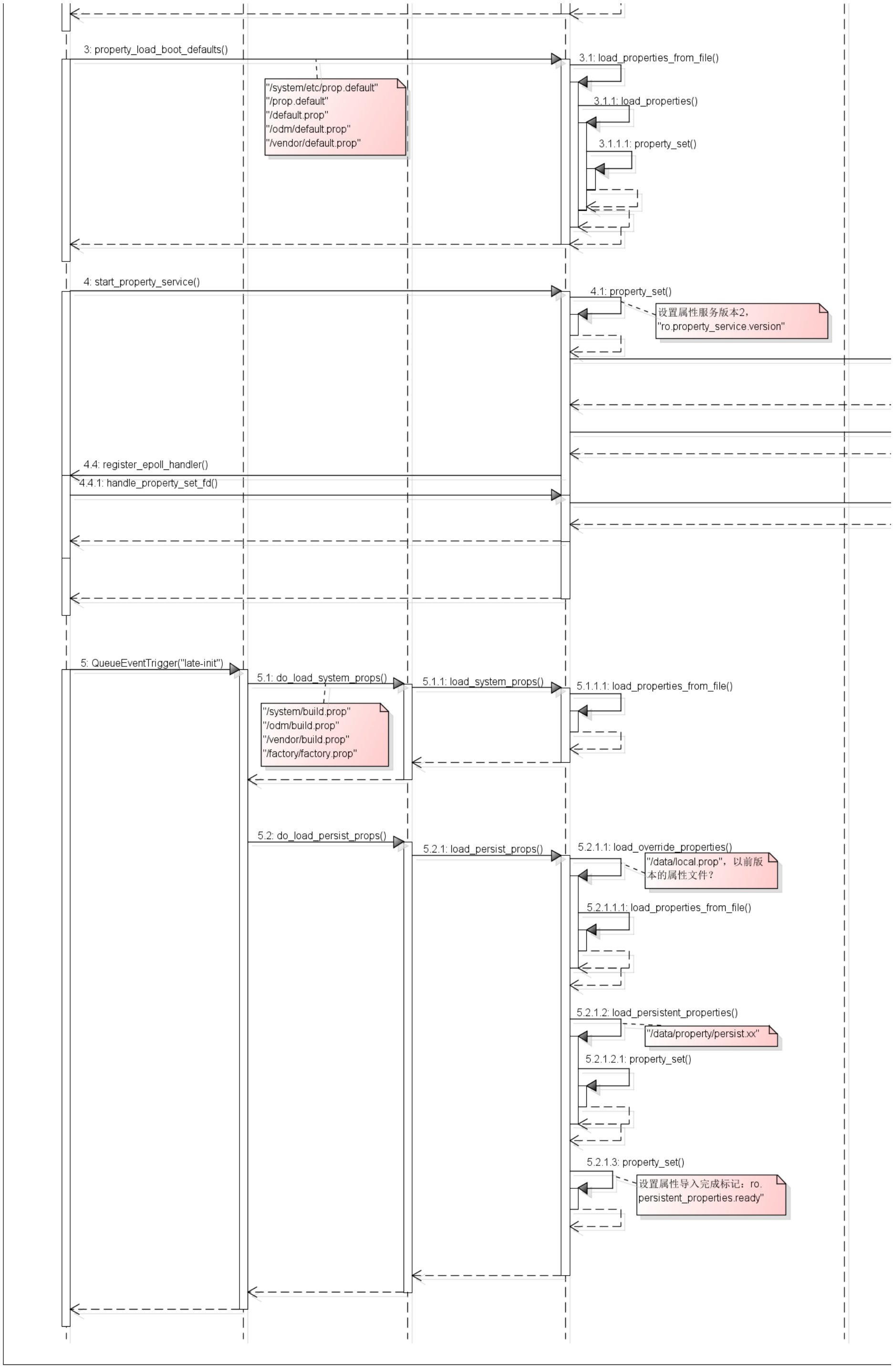
2.5. 加载系统属性和persist属性文件

- 在属性服务启动之后，实际保存在/data/property目录下的persist属性文件由于data分区未挂载尚未导入。
- 在data分区挂载完之后，init进程会继续触发动作导入系统属性文件和persist属性文件，系统属性文件目录如下：

```
"/system/build.prop"
"/odm/build.prop"
"/vendor/build.prop"
"/factory/factory.prop"
```

- 综上，Android系统属性服务的启动流程如下：





3. 属性设置

3.1. 设置接口的调用

- 当java进程设置属性时，首先调用SystemProperties.set(key，val)接口;
- 之后通过JNI调用到native层接口文件properties中的property_set();
- property_set()调用的是system_properties接口文件提供的__system_property_set()方法。

3.2. socket传递设置指令

- __system_property_set()方法中首先连接到"/dev/socket/property_service"套接字。
- 然后按照属性服务的协议，向套接字发送协议版本号、属性名、属性值。

3.3. 属性服务处理指令

- 属性服务调用property_service.cpp文件的handle_property_set_fd()中监听到属性设置请求，读出相关信息后调用handle_property_set()进行处理。

3.3.1. 对于控制指令的处理

- handle_property_set()函数中首先调用is_legal_property_name()判断属性名是否合规，然后调用StartsWith()判断属性是不是“ctl.xx”控制指令。
- 如果确认是“ctl.xx”控制属性，并且selinux权限检查通过，则调用handle_control_message()执行启动、停止、重启服务的操作。
- “ctl.xx”可以控制启动、停止、重启由init进程启动的服务（.rc文件里定义的），可以在代码中调用，也可以通过shell直接控制，用法如下表：

接口	注释
setprop ctl.start service-name	启动一个服务（.rc 文件中的service）
setprop ctl.stop service-name	关闭一个服务 （.rc 文件中的service）
setprop ctl.restart service-name	重启一个服务 （.rc 文件中的service）

3.3.2. 对于普通属性的处理

- 如果是“ctl.xx”控制指令，执行完后会直接返回，相关属性指令不会加载到共享内存当中。
- 对于其他属性值，则在selinux安全检查通过之后，会调用property_service.cpp文件中的property_set()进行属性设置。
- property_set()方法中首先通过调用__system_property_find()在共享内存中查找属性是否存在。
- 对于已经存在的属性，调用__system_property_update()进行更新。
- 如果属性不存在，则调用__system_property_add()方法插入到共享内存当中。

3.3.3. 写persist属性文件

- 对于persist属性，在加载到共享内存之后，会判断init进程启动属性服务后的persist属性文件是否导入成功，如果成功则调用write_persistent_property()在/data/property目录下创建一个对于属性名称的属性文件，内容为属性值。

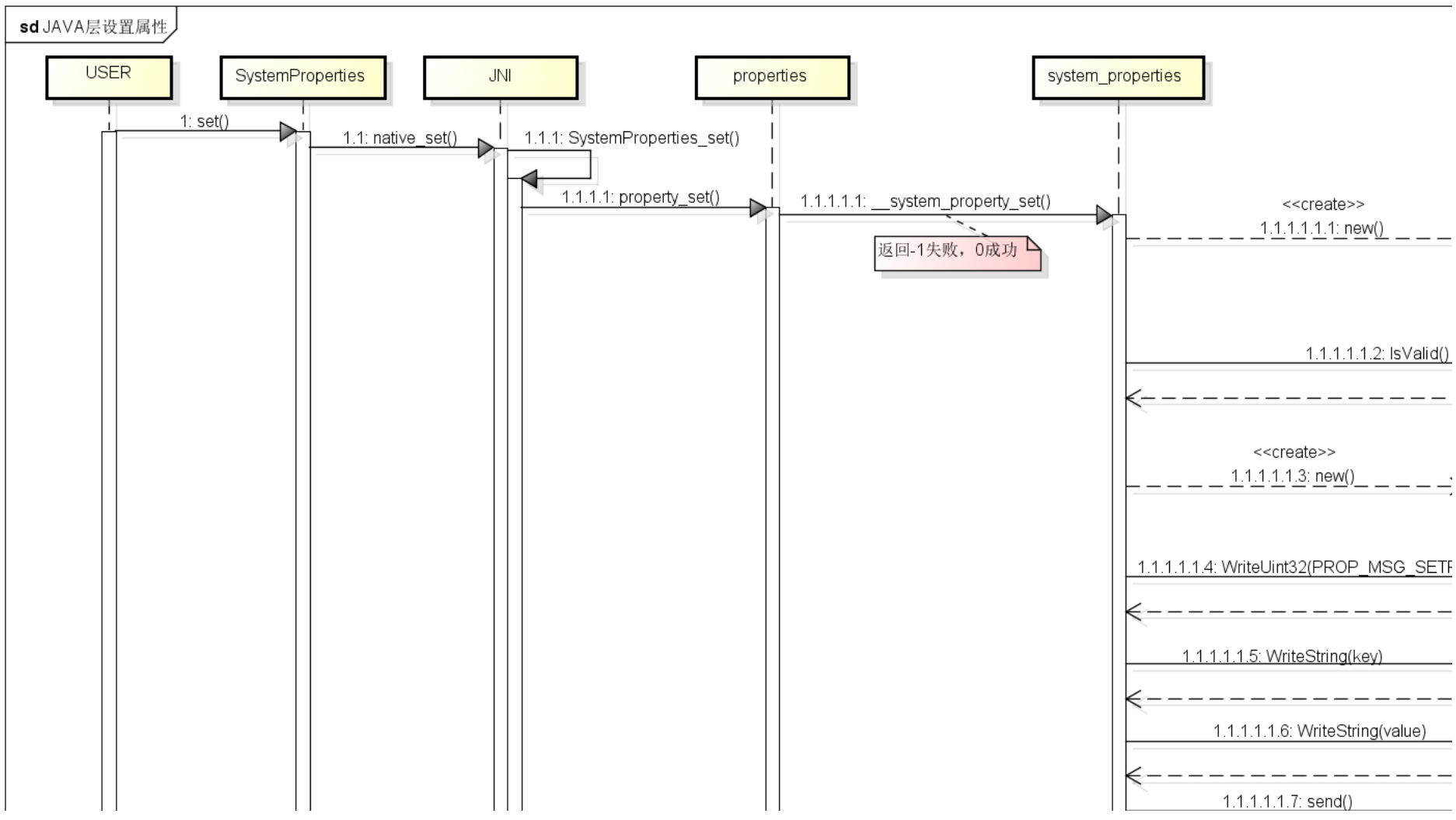
3.3.4. 通知init进程属性改变

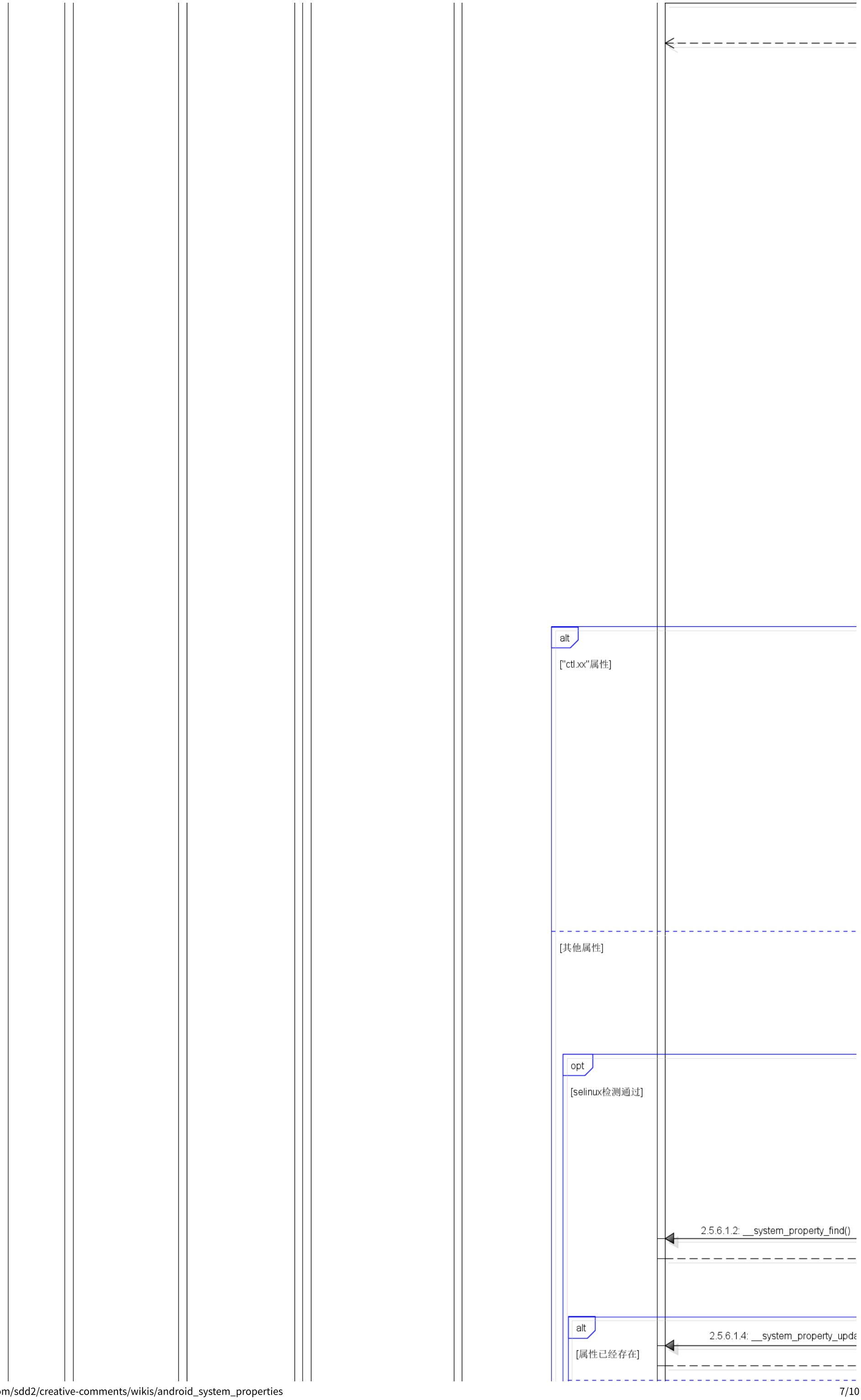
- 在普通属性的上述流程走完之后，property_set()方法还会调用一次property_changed()来通知init进程的主线程相关属性已经改变了。
- 如果属性名称和属性值正好和.rc文件中定义的action一致，则会触发init进程的主线程执行此action。例如logcatd.rc中定义了"persist.logd.logpersistd.enable=true"和"logd.logpersistd=logcatd"的相关action，如下：

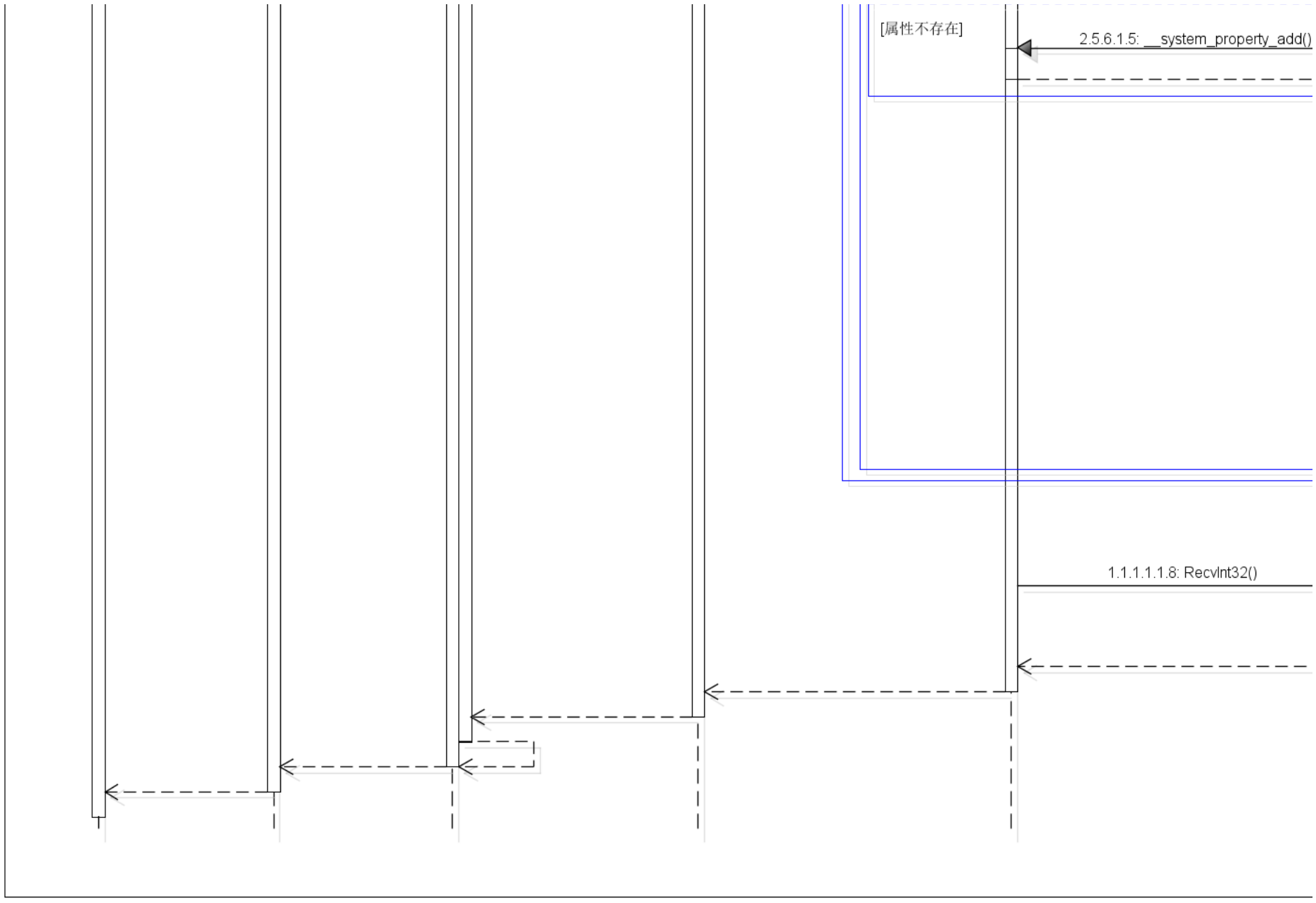
```
 # enable, prep and start logcatd service
on load_persist_props_action
      setprop persist.logd.logpersistd.enable true

on property:persist.logd.logpersistd.enable=true &&  property:logd.logpersistd=logcatd
     mkdir /data/misc/logd 0700 logd log
     start logcatd
```

3.3.5. selinux安全检测

- 在eng版本下，selinux的宽容模式不会对没有权限的客户端进行实际的拦截，只会提示权限不足的log。
- 在user版本下，selinux安全机制会执行拦截操作，因此应用时前需按selinux的拦截规制添加相应的权限，对此本文不展开。
- 以Java层应用为例，设置属性的时序如下:




alt
["ctl.xx"属性]
[其他属性]
opt
[selinux检测通过]
2.5.6.1.2: __system_property_find()
alt
[属性已经存在]
2.5.6.1.4: __system_property_upda



4. 属性读取

4.1. 读取接口的调用

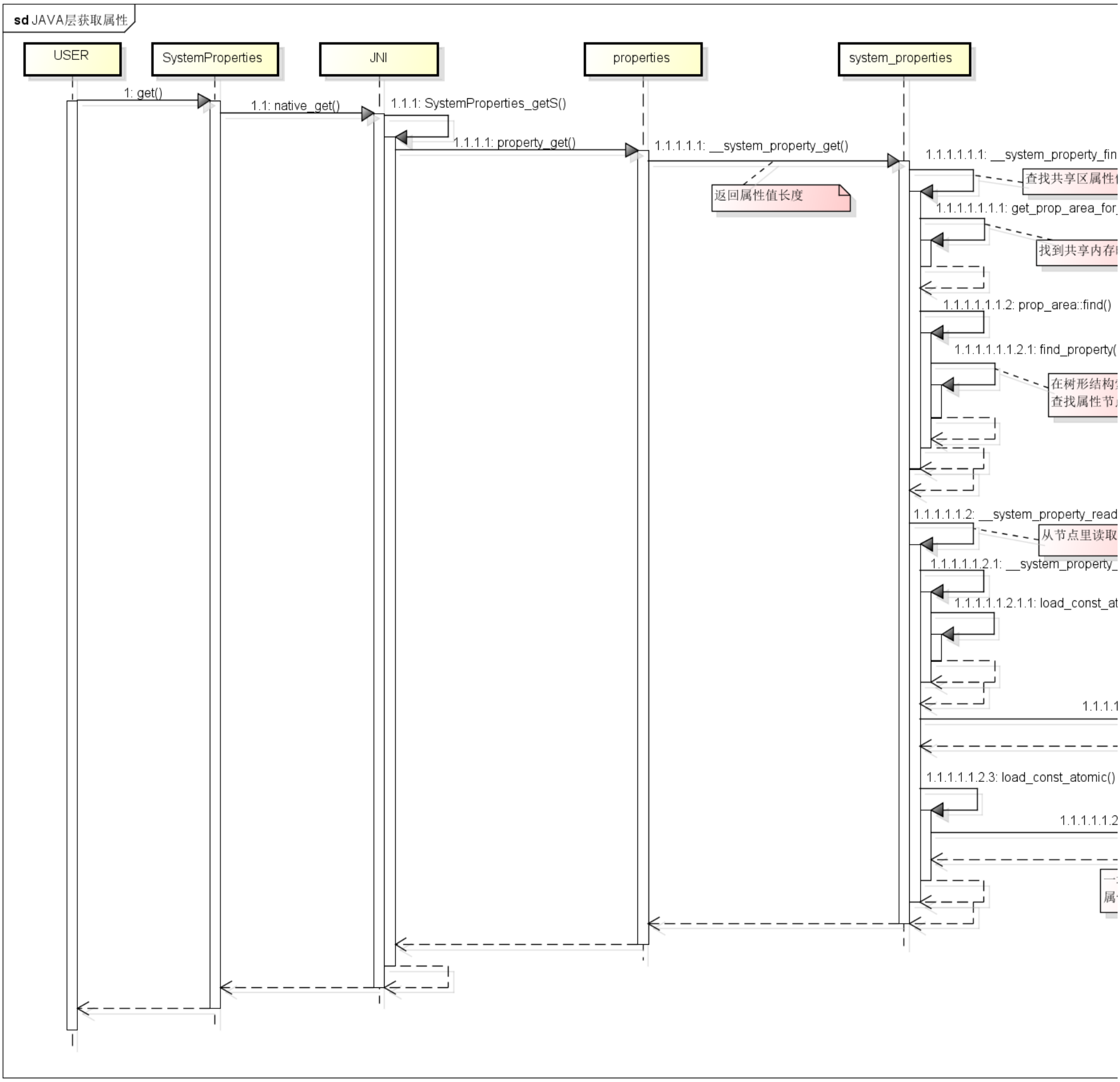
- 在Java层和Native层封装很多获取属性获取接口，但在共享内存中的属性都是以字符串形式保存的，因此各种接口的返回值也都是从字符串转换而来的。
- 以Java进程为例，获取指定属性的属性值时，可以调用SystemProperties.get(key)接口。
- 之后通过JNI调用到native层接口文件properties中的property_get();
- property_get()调用的是system_properties接口文件提供的__system_property_get()方法。
- __system_property_get()中的操作分为两步，分别调用的是_system_property_find()查找属性节点和_system_property_read()从节点中读出属性值。

4.2. 共享区查找属性节点

- __system_property_find()方法中，调用get_prop_area_for_name()获取共享内存的描述符，然后通过find_property()方法按字典树的规则查找到对应属性名的节点。

4.3. 共享区读取属性值

- __system_property_read()方法则是从_system_property_find()找到的数据节点中读取属性值，然后返回。
- 以Java层应用为例，获取属性的时序如下:

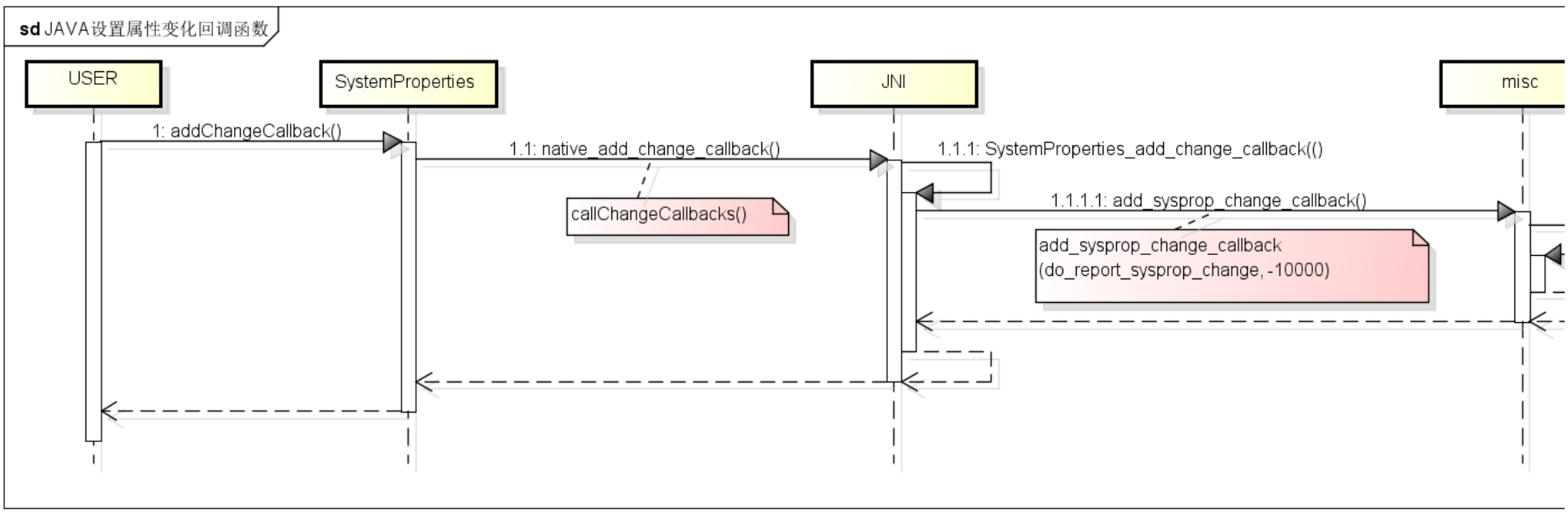


5. *JAVA层属性特殊应用

- 在第一节介绍java接口时，有两个不常用的接口，添加回调和报告属性变化。从代码上看这两个和共享内存中的属性变化似乎没有什么关系，但却放在同一个类中。避免误解其作用，这里单独进行分析。

5.1. 设置添加回调函数

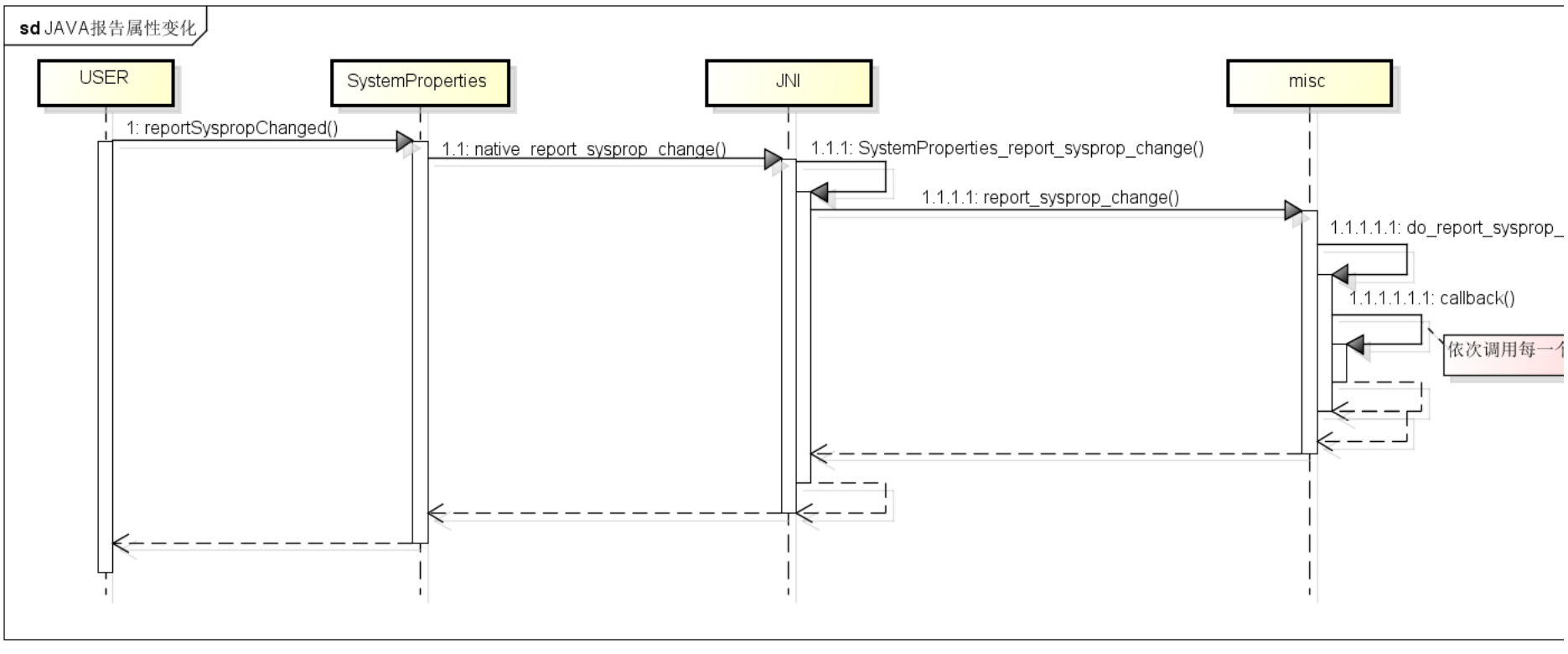
- 应用调用addChangeCallback()接口后，在JNI文件android_os_SystemProperties.cpp中又调用了misc.cpp中的add_sysprop_change_callback()，将回调函数添加进了misc.cpp文件中的一个静态的list中。时序如下图：



- 整个流程没有关联到Android系统属性的共享内存。

5.2. 通知执行属性回调

- reportSyspropChanged()接口没传递什么参数，字面理解是Java应用主动报告系统属性变化时调用。其最终调用到misc.cpp的report_sysprop_change()函数，并在该函数中依次调用了list里面存在的回调函数。也就是说，报告一次会通知所有的回调。时序图如下：



5.3. *JAVA系统属性机制

- 在JAVA层应用程序中，除了可以通过SystemProperties类设置获取Native层的Android系统属性之外，也可以通过System类调用System.getProperty和System.setProperty方法来设置和获取JAVA层自己的属性。但需要注意的是这两种属性机制是不通用的，不能从JAVA属性System类的接口读取Android系统属性值。
- JAVA属性运行在虚拟机之上，可以获取操作系统的类型、用户JDK版本、用户工作目录等随工作平台变化的信息等信息，设置和获取接口与Android属性类似。这里不再展开，需要了解的可以参考下 java之系统属性、 JAVA属性列表。

6. Android系统属性的局限

- 总结来说，Android系统属性功能强大，主要包括优点：
  - 应用范围广，在Java层、native层、shell层均可获取属性，只要权限足够，都可以发送指令通过init修改属性;
  - 由init进程管理，在大多数进程启动之前完成初始化，因此可以参与大多数进程启动时的配置;
  - 接口简单，get/set();
  - 可以保存属性文件，配置永久属性。
- 虽然功能强大，但在实际操作中，Android系统属性有一定的不足与局限：
  - 裁减了UID权限检查，那么在eng版下的任何用户都可以修改属性；而user版本使用，需要设置对应的selinux权限；
  - 共享内存中新增的属性无法删除；
  - 共享内存大小只有128kb，容量有限，超出后就无法再添加属性；
  - 只能以字符串形式保存一些简短的信息，作用有限；
  - 只有persist属性是永久保存的，但是persist属性保存在/data分区，导致启动的时候，一些在/data分区挂载之前启动的进程不能及时获取和保存persist属性。