

资讯 | 安全 | 论坛 | 下载 | 读书 | 程序开发 | 数据库 | 系统 | 网络 | 电子书 | 微信学院 | 站长学院 | QQ | 手机软件 | 考试

软件开发 | web前端 | Web开发 | 移动开发 | 综合编程 | [登录](#) | [注册](#)



短信接口api



pcb抄板



上海酒店式公寓



手持式显微镜



android视频教程



千峰培训



红外热成像仪



app外包

首页 > 程序开发 > 移动开发 > Android > 正文

Android Camera API2中采用CameraMetadata用于从APP到HAL的参数交互

2015-10-30

0 条评论

来源：天才之嵌入式

收藏

[我要投稿](#)



[android视频教程](#) [app外包](#) [骁龙cpu排行](#) [红外热成像仪](#) [app制作教程](#) [液压升降平台](#)

[app开发制作](#) [大疆无人机](#) [无人机培训](#) [上海酒店式公寓](#) [移动无限流量卡](#)

前沿：

在全新的Camera API2架构下，常常会有人疑问再也看不到熟悉的SetParameter/Paramters等相关的身影，取而代之的是一种全新的CameraMetadata结构的出现，他不仅很早就出现在Camera API1/API2结构下的Camera2Device、Camera3Device中用于和HAL3的数据交互，而现在在API2的驱使下都取代了Parameter，实现了Java到native到hal3的参数传递。那么现在假如需要在APP中设置某一项控制参数，对于Camera API2而言，涉及到对Sensor相关参数的set/control时又需要做哪些工作呢？

1. camera_metadata类整体布局结构

主要涉及到的源文件包括camera_metadata_tags.h，camera_metadata_tag_info.c，CameraMetadata.cpp，camera_metadata.c。对于每个Metadata数据，其通过不同业务控制需求，将整个camera工作需要的参数划分成多个不同的Section，其中在camera_metadata_tag_info.c表定义了所有Camera需要使用到的Section段的Name：

```
const char *camera_metadata_section_names[ANDROID_SECTION_COUNT] = {  
  
    [ANDROID_COLOR_CORRECTION]    = android.colorCorrection,  
  
    [ANDROID_CONTROL]              = android.control,  
  
    [ANDROID_DEMOSAIC]             = android.demosaic,  
  
    [ANDROID_EDGE]                 = android.edge,  
  
    [ANDROID_FLASH]                = android.flash,  
  
    [ANDROID_FLASH_INFO]           = android.flash.info,  
  
    [ANDROID_GEOMETRIC]            = android.geometric,  
  
    [ANDROID_HOT_PIXEL]            = android.hotPixel,
```



嵌入式主板



文章

读书

· Win2000下关闭无用端口

· 禁止非法用户登录综合设置 [win9x篇]

· 关上可恶的后门——消除NetBIOS隐患

· 网络入侵检测系统

· 潜伏在Windows默认设置中的陷阱

· 调制解调器的不安全

· 构建Windows 2000服务器的安全防护林

· SQL Server 2000的安全配置

点击排行

· Android N开发

· Android逆向之旅---动态方式破解apk进

· Android_03_获取数据库信息并显示在界

· RecyclerView完全解析,让你从此爱上它


· [Android Studio 权威教程] 断点调

· android 利用socket 发送Json数据demo


· Android M新控件之AppBarLayout, Nav

· 打造浪漫的Android表白程序


今日头条



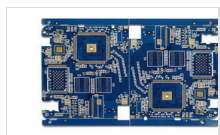
软件工程师待遇



android视频教程



手持式显微镜



pcb抄板

新闻排行榜

天

```
[ANDROID_HOT_PIXEL_INFO]      = android.hotPixel.info,
[ANDROID_JPEG]                = android.jpeg,
[ANDROID_LENS]                = android.lens,
[ANDROID_LENS_INFO]           = android.lens.info,
[ANDROID_NOISE_REDUCTION]     = android.noiseReduction,
[ANDROID_QUIRKS]              = android.quirks,
[ANDROID_REQUEST]             = android.request,
[ANDROID_SCALER]              = android.scaler,
[ANDROID_SENSOR]              = android.sensor,
[ANDROID_SENSOR_INFO]         = android.sensor.info,
[ANDROID_SHADING]             = android.shading,
[ANDROID_STATISTICS]          = android.statistics,
[ANDROID_STATISTICS_INFO]     = android.statistics.info,
[ANDROID_TONEMAP]             = android.tonemap,
[ANDROID_LED]                 = android.led,
[ANDROID_INFO]                = android.info,
[ANDROID_BLACK_LEVEL]         = android.blackLevel,
};
```

对于每个Section端而言，其都占据一个索引区域section_bounds，比如ANDROID_CONTROL Section他所代表的control区域是从ANDROID_CONTROL_START到ANDROID_CONTROL_END之间，且每个Section所拥有的Index范围理论最大可到(1 << 16)大小，完全可以满足统一Section下不同的控制参数的维护。

以ANDROID_CONTROL为列，他的Section index = 1，即对应的section index区间可到（1<<16，2<<16），但一般以实际section中维护的tag的数量来结束，即ANDROID_CONTROL_END决定最终的section index区间。对于每一个section，其下具备不同数量的tag，这个tag是一个指定section下的index值，通过该值来维护一个tag所在的数据区域，此外每个tag都有相应的string name，在camera_metadata_tag_info.c通过struct tag_info_t来维护一个tag的相关属性：

```
typedef struct tag_info {
    const char *tag_name;
    uint8_t     tag_type;
} tag_info_t;
```

其中tag_name为对应section下不同tag的name值，tag_type指定了这个tag所维护的数据类型，包括如下：

```
enum {
    // Unsigned 8-bit integer (uint8_t)
    TYPE_BYTE = 0,
    // Signed 32-bit integer (int32_t)
    TYPE_INT32 = 1,
```

- 1 Win10小马 原版镜像激活工具 永
- 2 KMS通用激活工具v2016.05.
- 3 Microsoft Toolkit (
- 4 C++ QT库开发
- 5 基于Android的计步器(Pedo
- 6 FlashFXP Portable
- 7 我來給你松松土澆澆花
- 8 Android性能优化
- 9 win7永久激活码免费分享
- 10 DesktopOK (64

JD.COM 京东

¥ 1880.00

1/6

```
// 32-bit float (float)
TYPE_FLOAT = 2,

// Signed 64-bit integer (int64_t)
TYPE_INT64 = 3,

// 64-bit float (double)
TYPE_DOUBLE = 4,

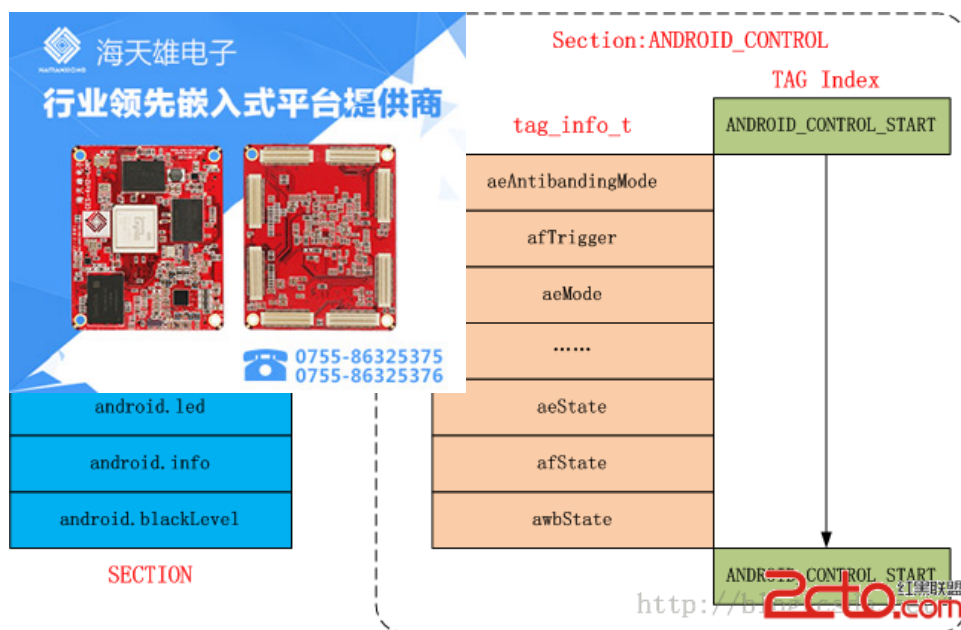
// A 64-bit fraction (camera_metadata_rational_t)
TYPE_RATIONAL = 5,

// Number of type fields
NUM_TYPES
};
```

对每一个section所拥有的tag_info信息，通过全局结构体tag_info_t

*tag_info[ANDROID_SECTION_COUNT] 来定义。

下图是对整个Camera Metadata对不同section以及相应section下不同tag的布局图，下图以最常见的android.control Section为例进行了描述：



2. CameraMetadata通过camera_metadata来维护数据信息

假设现在存在一个CameraMetadata对象，那么他是如何将一个tag标记的参数维护起来的呢？

```

CameraMetadata::CameraMetadata(size_t entryCapacity, size_t dataCapacity) :
    mLocked(false)
{
    mBuffer = allocate_camera_metadata(entryCapacity, dataCapacity);
}

camera_metadata_t *allocate_camera_metadata(size_t entry_capacity,
                                           size_t data_capacity) {
    if (entry_capacity == 0) return NULL;

    size_t memory_needed = calculate_camera_metadata_size(entry_capacity,
                                                         data_capacity);

    void *buffer = malloc(memory_needed);

    return place_camera_metadata(buffer, memory_needed,
                                entry_capacity,
                                data_capacity);
}

```

一个CameraMetadata数据内存块中组成的最小基本单元是struct camera_metadata_buffer_entry，总的entry数目等信息需要struct camera_metadata_t来维护：

```

struct camera_metadata {
    size_t          size;
    uint32_t        version;
    uint32_t        flags;
    size_t          entry_count; // 当前实际的entry数目
    size_t          entry_capacity; // entry最大可以存储的数目
    uptrdiff_t      entries_start; // Offset from camera_metadata
    size_t          data_count; // 当前占据的数据空间
    size_t          data_capacity; // 最大可操作的数据容量
    uptrdiff_t      data_start; // Offset from camera_metadata, 大容量数据存储的起
    void            *user; // User set pointer, not copied with buffer
    uint8_t         reserved[0];
};

```

对于每一个entry主要记录他的所代表的TAG，以及这个TAG的需要存储的数据类型，此外还需要记录这个entry是否需要一个union offset来表示他当前数据量过大时的数据存储位置，

```

typedef struct camera_metadata_buffer_entry {
    uint32_t tag; // 表示当时这个entry代表的tag值，即上文提到的section中不同的tag index值

```

```

size_t    count;

union {

    size_t    offset;

    uint8_t value[4];

} data; //如果存储的数据量不大于4则直接存储。否则需要指点一个offset来表示便宜

uint8_t    type; //维护的数据类型

uint8_t    reserved[3];

} camera_metadata_buffer_entry_t;

```



3. update更新并建立参数

CameraMetadata支持不同类型的数据更新或者保存到camera_metadata_t中tag所在的entry当中去，以一个更新单字节的数据为例，data_count指定了数据的个数，而tag指定了要更新的entry。

```

status_t CameraMetadata::update(uint32_t tag,
    const uint8_t *data, size_t data_count) {

    status_t res;

    if (mLocked) {

        ALOGE("%s: CameraMetadata is locked, __FUNCTION__");

        return INVALID_OPERATION;

    }

    if ( (res = checkType(tag, TYPE_BYTE)) != OK) {

        return res;

    }

    return updateImpl(tag, (const void*)data, data_count);

}

```

首先是通过checkType，主要是通过tag找到get_camera_metadata_tag_type其所应当支持的tag_type(因为具体的TAG是已经通过camera_metadata_tag_info.c源文件中的tag_info这个表指定了其应该具备的tag_type)，比较两者是否一致，一致后才允许后续的操作，如这里需要TYPE_BYTE一致。

updateImpl函数主要是讲所有要写入的数据进行update操作。

```
status_t CameraMetadata::updateImpl(uint32_t tag, const void *data,
    size_t data_count) {
    status_t res;
    if (mLocked) {
        ALOGE("%s: CameraMetadata is locked, __FUNCTION__");
        return INVALID_OPERATION;
    }
    int type = get_camera_metadata_tag_type(tag);
    if (type == -1) {
        ALOGE("%s: Tag %d not found, __FUNCTION__, tag);
        return BAD_VALUE;
    }
    size_t data_size = calculate_camera_metadata_entry_data_size(type,
        data_count);

    res = resizeIfNeeded(1, data_size); //新建camera_metadata_t

    if (res == OK) {
        camera_metadata_entry_t entry;
        res = find_camera_metadata_entry(mBuffer, tag, &entry);
        if (res == NAME_NOT_FOUND) {
            res = add_camera_metadata_entry(mBuffer,
                tag, data, data_count); //将当前新的tag以及数据加入到camera_metadata_t
        } else if (res == OK) {
            res = update_camera_metadata_entry(mBuffer,
                entry.index, data, data_count, NULL);
        }
    }

    if (res != OK) {
        ALOGE("%s: Unable to update metadata entry %s.%s (%x): %s (%d),
            __FUNCTION__, get_camera_metadata_section_name(tag),
            get_camera_metadata_tag_name(tag), tag, strerror(-res), res);
    }

    IF ALOGV() {
        ALOGE_IF(validate_camera_metadata_structure(mBuffer, /*size*/NULL) !=
```

```

        OK,

        %s: Failed to validate metadata structure after update %p,
        __FUNCTION__, mBuffer);
    }

    return res;
}

```

主要分为以下几个过程：

- a.通过tag_type存储的数据类型，由calculate_camera_metadata_entry_data_size计算要写入的entry中的数据量。
- b.resizeIfNeeded通过已有entry的数量等，增加entry_capacity，或者重建整个camera_metadata_t，为后续增加数据创建内存空间基础。
- c.通过find_camera_metadata_entry获取一个entry的入口camera_metadata_entry_t，如果存在这个tag对应的entry，则将camera_metadata_buffer_entry_t的属性信息转为camera_metadata_entry_t。

```

typedef struct camera_metadata_entry {
    size_t    index; //在当前的entry排序中，其所在的index值

    uint32_t tag;

    uint8_t type;

    size_t    count;

    union {
        uint8_t *u8;
        int32_t *i32;
        float    *f;
        int64_t *i64;
        double   *d;
        camera_metadata_rational_t *r;
    } data; //针对不同数据类型，u8表示数据存储的入口地址，不大于4字节即为value[4]。
} camera_metadata_entry_t;

```

d.add_camera_metadata_entry完成全新的entry更新与写入，即这个TAG目前不存在于这个camera_metadata_t中；update_camera_metadata_entry则是直接完成数据的更新。

3. Java层中CameraMetadata.java和CameraMetadataNative.java

下面以API2中java层中设置AF的工作模式为例，来说明这个参数设置的过程：

```
mPreviewBuilder.set(CaptureRequest.CONTROL_AF_MODE, CaptureRequest.CONTROL_AF_MODE_CONTI
```

其中CONTROL_AF_MODE定义在CaptureRequest.java中如下以一个Key的形式存在：

```
public static final Key CONTROL_AF_MODE =  
    new Key(android.control.afMode, int.class);  
  
public Key(String name, Class type) {  
    mKey = new CameraMetadataNative.Key(name, type);  
}
```

在CameraMetadataNative.java中Key的构造

```
public Key(String name, Class type) {  
    if (name == null) {  
        throw new NullPointerException(Key needs a valid name);  
    } else if (type == null) {  
        throw new NullPointerException(Type needs to be non-null);  
    }  
    mName = name;  
    mType = type;  
    mTypeReference = TypeReference.createSpecializedTypeReference(type);  
    mHash = mName.hashCode() ^ mTypeReference.hashCode();  
}
```

其中CONTROL_AF_MODE_CONTINUOUS_PICTURE定义在CameraMetadata.java中

```
public static final int CONTROL_AF_MODE_CONTINUOUS_PICTURE = 4;
```

逐一定位set的入口:

a. mPreviewBuilder是CaptureRequest.java的build类, 其会构建一个CaptureRequest

```
public Builder(CameraMetadataNative template) {  
    mRequest = new CaptureRequest(template);  
}
```



```
private CaptureRequest() {  
    mSettings = new CameraMetadataNative();  
    mSurfaceSet = new HashSet();  
}
```

mSetting建立的是一个CameraMetadataNative对象，主要用于和Native层进行接口交互，构造如下

```
public CameraMetadataNative() {  
    super();  
    mMetadataPtr = nativeAllocate();  
    if (mMetadataPtr == 0) {  
        throw new OutOfMemoryError(Failed to allocate native CameraMetadata);  
    }  
}
```

b. CaptureRequest.Build.set()

```
public void set(Key key, T value) {  
    mRequest.mSettings.set(key, value);  
}  
  
public void set(CaptureRequest.Key key, T value) {  
    set(key.getNativeKey(), value);  
}
```

考虑到CaptureRequest extend CameraMetadata，则CaptureRequest.java中getNativeKey

```
public CameraMetadataNative.Key getNativeKey() {  
    return mKey;  
}
```

mKey即为之前构造的CameraMetadataNative.Key.

```
public void set(Key key, T value) {  
    SetCommand s = sSetCommandMap.get(key);  
    if (s != null) {  
        s.setValue(this, value);  
    }
```

```
        return;
    }

    setBase(key, value);
}

private void setBase(Key key, T value) {
    int tag = key.getTag();

    if (value == null) {
        // Erase the entry
        writeValues(tag, /*src*/null);
        return;
    } // else update the entry to a new value

    Marshaller marshaller = getMarshallerForKey(key);
    int size = marshaller.calculateMarshalSize(value);

    // TODO: Optimization. Cache the byte[] and reuse if the size is big enough.
    byte[] values = new byte[size];

    ByteBuffer buffer = ByteBuffer.wrap(values).order(ByteOrder.nativeOrder());
    marshaller.marshal(value, buffer);

    writeValues(tag, values);
}
```

首先来看key.getTag()函数的实现，他是将这个key交由Native层后转为一个真正的在Java层中的tag值：

```
public final int getTag() {
    if (!mHasTag) {
        mTag = CameraMetadataNative.getTag(mName);
        mHasTag = true;
    }
    return mTag;
}

public static int getTag(String key) {
    return nativeGetTagFromKey(key);
}
```

是将Java层的String交由Native来转为一个Java层的tag值。

再来看writeValues的实现，同样调用的是一个native接口，很好的阐明了CameraMetadataNative的意思：

```
public void writeValues(int tag, byte[] src) {  
    nativeWriteValues(tag, src);  
}
```

相关native层的实现在下一小节说明。

4. Native层的CameraMetadata结构完成camera参数的传递

在描述完了CameraMetadata数据的相关操作之后，可明确的一点是SECTION下的TAG是操作他的核心所在。

这里先说明一个在API1 Camera2Client 参数传递的过程，他采用的逻辑是还是在Java层预留了setParameters接口，只是当Parameter在设置时比起CameraClient而言，他是将这个Parameter根据不同的TAG形式直接绑定到CameraMetadata mPreviewRequest/mRecordRequest/mCaptureRequest中，这些数据会由Capture_Request转为camera3_capture_request中的camera_metadata_t settings完成参数从Java到native到HAL3的传递。

但是在Camera API2下，不再需要那么复杂的转换过程，在Java层中直接对参数进行设置并将其封装到Capture_Request即可，即参数控制由Java层来完成。这也体现了API2中Request和Result在APP中就大量存在的原因。对此为了和Framework Native层相关TAG数据的统一，在Java层中大量出现的参数设置是通过Section Tag的name来交由Native完成转换生成在Java层的TAG。

对于第三小节中提到的native层的实现，其对应的实现函数位于android_hardware_camera2_CameraMetadata.c中，如CameraMetadata_getTagFromKey是实现将一个Java层的string转为一个tag的值，他的主要原理如下：根据传入的key string值本质是由一个字符串组成的如上文中提到的android.control.mode。对比最初不同的Section name就可以发现前面两个x.y的字符串就是代表是Section name.而后面mode即是在该section下的tag数值，所以通过对这个string的分析可知，就可以定位他的section以及tag值。这样返回到Java层的就是key相应的tag值了。

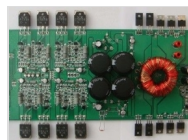
如果要写数据，那么在native同样需要一个CameraMetadata对象，这里是在Java构造CameraMetadataNative时实现的，调用的native接口是nativeAllocate()：

```
static jlong CameraMetadata_allocate(JNIEnv *env, jobject thiz) {  
    ALOGV("%s, __FUNCTION__");  
  
    return reinterpret_cast(new CameraMetadata());  
}
```

最终可以明确的是CameraMetadata相关的参数是被Java层来set/get，但本质是在native层进行了实现，后续如果相关控制参数是被打包到CaptureRequest中时传入到native时即操作的还是native中的CameraMetadata。



短信接口api



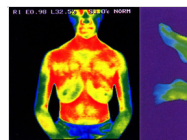
pcb抄板



android视频教程



手持式显微镜



红外热成像仪

[点击复制链接 与好友分享!](#)[回本站首页](#)[上一篇: android-仿iOS弹出框](#)[下一篇: AndroidManifest配置文件介绍](#)

相关文章

[activity之间参数传递](#)[共享参数ContentProvider 类与数据库](#)[onCreate \(\) 方法中的参数Bundle save](#)[Android采用SharedPreferences保存用户](#)[Android Button setTextColor\(\)参数](#)[Android 访问Webservice接口, 参数对](#)[android屏幕分类与屏幕相关参数定义](#)[Android 批量设置监听器, 监听器传递](#)[Android客户端性能参数监控](#)[Android 开发之 Activity 状态保存](#)

力动Rido健身车家用动感单车静音 B5健身车
¥1880.00



牧树人2016春夏装新款 免烫短袖衬衫男 商务
¥178.00

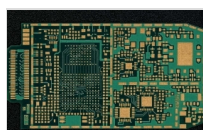


鞋狂男鞋休闲鞋男潮流板鞋透气舒适简约百搭
¥99.00

图文推荐



android视频教程



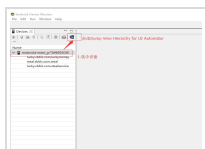
pcb抄板



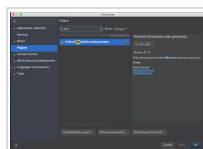
短信接口api



千峰培训



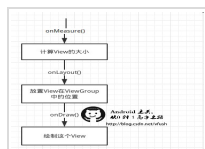
从抢红包插件谈Access



android产品研发 (十



Android官方文档之Ap



Android 面试题总结

我有话说(0条评论)



来说两句吧...

[搜狐登录](#)[微博登录](#)[QQ登录](#)[手机登录](#)


还没有评论, 快来抢沙发吧!

红黑联盟正在使用畅言

pcb抄板 短信接口api 千峰培训
android视频教程 android开发入门

app外包 手持式显微镜 大疆无人机
骁龙cpu排行 监控摄像头价格

上海酒店式公寓 app开发制作
app界面设计 红外热成像仪

**证书 + 能力**

安全工程师 软件工程师 网站工程师 网络工程师 电脑工程师
为新手量身定做的课程，让菜鸟快速变身高手 正规公司助您腾飞

不断增加新科目
立即加入

[关于我们](#) | [联系我们](#) | [广告服务](#) | [投资合作](#) | [版权申明](#) | [在线帮助](#) | [网站地图](#) | [作品发布](#) | **Vip技术培训**

版权所有: 红黑联盟--致力于做最好的IT技术学习网站