NXP ZigBee3.0 协议栈消息处理流程分析

(Shaozhong.Liang)

NXP ZigBee3.0SDK 相比之前的版本有较大的改进，简化了 JenOS 操作系统，采用更容易理解的前后台方式。我们以 ZigBee3.0 例程 JN-AN-1218-Zigbee-3-0-Light-Bulb 为例子，分析收到 On/Off Cluster(0x0006)消息时的函数调用流程。

在程序启动后调用代码入口 vAppMain 函数，初始化 Cluster Instance 实例，并注册 EndPoint 回调函数的过程如下：

vAppMain

→vInitialiseApp

→ APP_ZCL_vInitialise

→eApp_ZLO_RegisterEndpoint

→eZLO_RegisterDimmableLightEndPoint

→eCLD_OnOffCreateOnOff

```
PUBLIC void APP_ZCL_vInitialise(void)
{
    ..........................

    /* Initialise ZLL */
    eZCL_Status = eZCL_Initialise(&APP_ZCL_cbGeneralCallback, apduZCL);
    if (eZCL_Status != E_ZCL_SUCCESS)
    {
        DBG_vPrintf(TRACE_ZCL, "\nErr: eZLO_Initialise:%d", eZCL_Status);
    }

    ..........................

    /* Register Light EndPoint */
    eZCL_Status = eApp_ZLO_RegisterEndpoint(&APP_ZCL_cbEndpointCallback);
    if (eZCL_Status != E_ZCL_SUCCESS)
    {
        DBG_vPrintf(TRACE_ZCL, "Error: eZLL_RegisterCommissionEndPoint:%d\r\n", eZCL_Status);
    }
    ..........................
}

PUBLIC teZCL_Status eZLO_RegisterDimmableLightEndPoint(uint8 u8EndPointIdentifier,
                                                       tfpZCL_ZCLCallBackFunction cbCallBack,
                                                       tsZLO_DimmableLightDevice *psDeviceInfo)
{
    ........................

    #if (defined CLD_BASIC) && (defined BASIC_SERVER)
        /* Create an instance of a basic cluster as a server */
        if(eCLD_BasicCreateBasic(&psDeviceInfo->sClusterInstance.sBasicServer,
                                 TRUE,
                                 &sCLD_Basic,
                                 &psDeviceInfo->sBasicServerCluster,
                                 &au8BasicClusterAttributeControlBits[0]) != E_ZCL_SUCCESS)
        {
            // Need to convert from cluster specific to ZCL return type so we lose the extra inf
            return E_ZCL_FAIL;
        }
    #endif
    ..............................

    #if (defined CLD_ONOFF) && (defined ONOFF_SERVER)
        /* Create an instance of a On/Off cluster as a server */
        if(eCLD_OnOffCreateOnOff(&psDeviceInfo->sClusterInstance.sOnOffServer,
                                 TRUE,
                                 &sCLD_OnOff,
                                 &psDeviceInfo->sOnOffServerCluster,
                                 &au8OnOffAttributeControlBits[0],
                                 &psDeviceInfo->sOnOffServerCustomDataStructure) != E_ZCL_SUCCESS)
        {
            // Need to convert from cluster specific to ZCL return type so we lose the extra inf
            return E_ZCL_FAIL;
        }
    #endif
    ........................
}
```

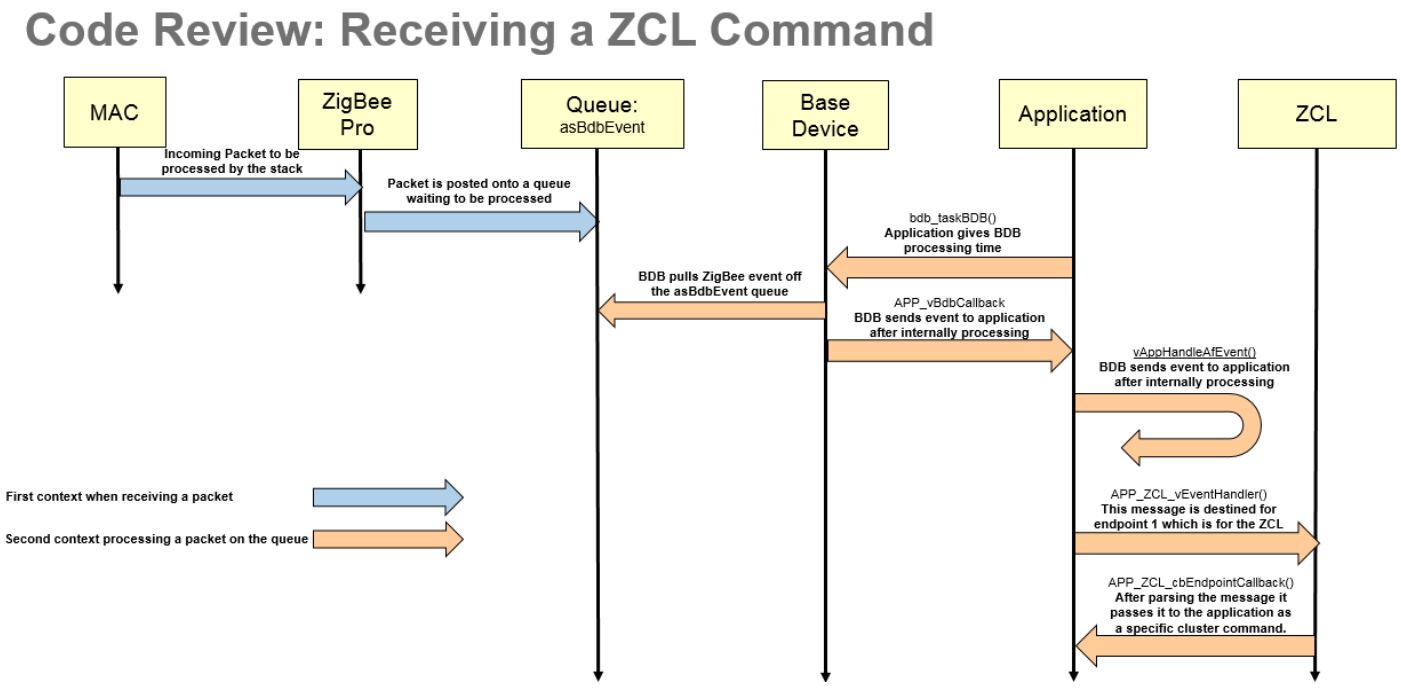
```
PUBLIC  teZCL_Status eCLD_OnOffCreateOnOff(
                tsZCL_ClusterInstance               *psClusterInstance,
                bool_t                               bIsServer,
                tsZCL_ClusterDefinition             *psClusterDefinition,
                void                                *pvEndPointSharedStructPtr,
                uint8                               *pu8AttributeControlBits,
                tsCLD_OnOffCustomDataStructure      *psCustomDataStructure)
{
    .................................

    // cluster data
    vZCL_InitializeClusterInstance(
           psClusterInstance,
           bIsServer,
           psClusterDefinition,
           pvEndPointSharedStructPtr,
           pu8AttributeControlBits,
           NULL,
           eCLD_OnOffCommandHandler);
    .................................

}
```

我们以 ZigBee3.0 例程 JN-AN-1218-Zigbee-3-0-Light-Bulb 为例子，分析收到 On/Off Cluster(0x0006)命令时的函数调用流程。

Ch.	Layer	Packet Information	MAC Src.	MAC Dst.
20	NWK	Beacon	0x838D	
20	ZCL	On/Off: On	0x0000	0x135C
20	ZCL	On/Off: On	0x0000	0x135C
20	ZCL	On/Off: On	0x0000	0x135C
20	MAC	Acknowledgement		
20	APS	Acknowledgement	0x135C	0x0000
20	MAC	Acknowledgement		
20	APS	Acknowledgement	0x135C	0x0000
20	MAC	Acknowledgement		
20	APS	Acknowledgement	0x135C	0x0000
20	MAC	Acknowledgement		
20	ZCL	On/Off: Default Resp…	0x135C	0x0000
20	MAC	Acknowledgement		
20	ZCL	On/Off: Report Attri…	0x135C	0x0000
20	MAC	Acknowledgement		
20	APS	Acknowledgement	0x0000	0x135C
20	MAC	Acknowledgement		
20	APS	Acknowledgement	0x0000	0x135C
20	MAC	Acknowledgement		

```
▷ MAC Header: (9 bytes)
◢ MAC Payload: (37 bytes)
  ▷ NWK Header: 0x621E0000135C0248
  ▷ NWK Aux Header: (14 bytes)
  ◢ NWK Payload: (11 bytes)
    ◢ APS Header: 0xA401010400060140
      ▷ Frame Control: 0x40
        Destination Endpoint: 0x01
        Cluster ID: [0x0006] General: On/Off
        Profile ID: [0x0104] ZigBee Home Automation
        Source Endpoint: 0x01
        APS Counter: 164
    ◢ APS Payload: 0x01AC01
      ◢ ZCL Header: 0x01AC01
        ◢ Frame Control: 0x01
            .... ..01 = Frame Type: [0x1] Command is Specific to a Cluster
            .... .0.. = Manufacturer Specific: [0x0] Manufacturer Code Not Included
            .... 0... = Direction: [0x0] From Client to Server
            ...0 .... = Disable Default Response: [0x0] No
            000. .... = Reserved: 0x0
          Transaction Sequence Number: 172
          Command ID: [0x01] On
```

下面 ZigBee 的处理流程图，任务 bdb_taskBDB 主要处理 ZigBee Cluster Library 相关的消息。



APP_vMainLoop
→bdb_taskBDB
→APP_vBdbCallback
→vAppHandleAfEvent
→APP_ZCL_vEventHandler
→vZCL_EventHandler
→vZCL_ZigbeeEventHandler

```
PRIVATE void vZCL_ZigbeeEventHandler(ZPS_tsAfEvent *pZPSevent)
{
   ......................

    // parse event structure
    switch(pZPSevent->eType)
    {
    case(ZPS_EVENT_APS_DATA_INDICATION):
        {
            if(pZPSevent->uEvent.sApsDataIndEvent.u8DstEndpoint != 0 &&
               pZPSevent->uEvent.sApsDataIndEvent.u8SrcEndpoint != 0 )
            {
                ..................................
                // handle ZCL message if success.  If error, PDU is not valid so pass up to user
                if (pZPSevent->uEvent.sApsDataIndEvent.eStatus == ZPS_E_SUCCESS)
                {
                    vZCL_HandleDataIndication(pZPSevent);
                }
                else
                {
                    sZCL_CallBackEvent.eEventType = E_ZCL_CBET_ERROR;
                    sZCL_CallBackEvent.eZCL_Status = E_ZCL_ERR_ZRECEIVE_FAIL;
                    psZCL_Common->eLastZpsError = pZPSevent->uEvent.sApsDataIndEvent.eStatus;
                    sZCL_CallBackEvent.u8EndPoint = pZPSevent->uEvent.sApsDataIndEvent.u8DstEndpoint;
                    vZCL_PassEventToUser(&sZCL_CallBackEvent);
                }
            }
            else
            {
                sZCL_CallBackEvent.eEventType = E_ZCL_CBET_UNHANDLED_EVENT;
                vZCL_PassEventToUser(&sZCL_CallBackEvent);
            }
            break;
        }
        ............................
}
```

深入分析 vZCL_HandleDataIndication 函数的处理过程。函数 eZCL_SearchForClusterEntry 通过 u16ClusterId 找到对应的 Cluster Instance 实例，并调用 Cluster 注册的回调函数。这个回调函数在 eZLO_RegisterDimmableLightEndPoint 初始化时作为参数保存在 Cluster Instance 的数据结构内存中。

```
PRIVATE void vZCL_HandleDataIndication(ZPS_tsAfEvent *pZPSevent)
{
    ............................
    if(eZCL_SearchForEPentryAndCheckManufacturerId(pZPSevent->uEvent.sApsDataIndEvent.u8DstEndpoi
    nt,      sZCL_HeaderParams.bManufacturerSpecific,      sZCL_HeaderParams.u16ManufacturerCode,
    &psZCL_EndPointDefinition) != E_ZCL_SUCCESS)
    {
        if ( sZCL_HeaderParams.eFrameType == eFRAME_TYPE_COMMAND_IS_SPECIFIC_TO_A_CLUSTER )
        {
            u8Error = E_ZCL_CMDS_UNSUP_MANUF_CLUSTER_COMMAND;
        }
        else
        {
            u8Error = E_ZCL_CMDS_UNSUP_MANUF_GENERAL_COMMAND;
        }
```

```
    ..............................
}

if (u8Error != 0)
{
    // send response if possible/required
    // Trac 6 - don't send default response from a default response.
    if (sZCL_HeaderParams.u8CommandIdentifier != E_ZCL_DEFAULT_RESPONSE ||
        sZCL_HeaderParams.eFrameType != eFRAME_TYPE_COMMAND_ACTS_ACCROSS_ENTIRE_PROFILE)
    {
        sZCL_CallBackEvent.eEventType = E_ZCL_CBET_ERROR;
        sZCL_CallBackEvent.eZCL_Status = E_ZCL_ERR_EP_UNKNOWN;
        sZCL_CallBackEvent.u8EndPoint = pZPSevent->uEvent.sApsDataIndEvent.u8DstEndpoint;
        vZCL_PassEventToUser(&sZCL_CallBackEvent);
        eZCL_SendDefaultResponse(pZPSevent, u8Error);
    }
    // free buffer and return
    PDUM_eAPduFreeAPduInstance(pZPSevent->uEvent.sApsDataIndEvent.hAPduInst);
    return;
}
    // check the command is suitable for the endpoint - cluster, manufac Id, direction
    eCallbackReturn = eZCL_SearchForClusterEntry(
                                            pZPSevent->uEvent.sApsDataIndEvent.u8DstEndpoint,
                                            pZPSevent->uEvent.sApsDataIndEvent.u16ClusterId,
                                            !sZCL_HeaderParams.bDirection,
                                            &psClusterInstance);
.........................

// Is command cluster specicfic or general
switch (sZCL_HeaderParams.eFrameType)
{
case eFRAME_TYPE_COMMAND_ACTS_ACCROSS_ENTIRE_PROFILE:
    {
        if (sZCL_HeaderParams.u8CommandIdentifier == E_ZCL_DEFAULT_RESPONSE)
        {
            // fill in callback event
            ……
            // call user directly
            psZCL_EndPointDefinition->pCallBackFunctions(&sZCL_CallBackEvent);
            break;
        }
        else if((pZPSevent != NULL) || \
                (psZCL_EndPointDefinition != NULL))
        {
            // check whether cluster is present on endpoint
            if ((psClusterInstance == NULL)
                && (bZCL_OverrideHandlingEntireProfileCmd(
                        pZPSevent->uEvent.sApsDataIndEvent.u16ClusterId) == FALSE))
            {
```

```
                eZCL_SendDefaultResponse(pZPSevent, E_ZCL_CMDS_UNSUPPORTED_CLUSTER);
            }
            else
            {
                // Moved to zcl_library_options.h as some commands are optional so
                // the command handler is built at the same time as the app and unused
                // optional commands are garbage collected.
                vZCL_HandleEntireProfileCommand(sZCL_HeaderParams.u8CommandIdentifier,
                    pZPSevent,
                    psZCL_EndPointDefinition,
                    psClusterInstance);
            }
        }
        break;
    }

case eFRAME_TYPE_COMMAND_IS_SPECIFIC_TO_A_CLUSTER:
    {
        // check whether cluster is present on endpoint
                eCallbackReturn = psClusterInstance->pCustomcallCallBackFunction(pZPSevent,
                                                                    psZCL_EndPointDefinition,
                                                                    psClusterInstance);
                ..................................
    }
    break;
default:
    {
        // Unknown frame type
        // Not doing a user call back here to save some code size as eFrameType
        // can't be out of range for a bad message
        eZCL_SendDefaultResponse(pZPSevent, E_ZCL_CMDS_SOFTWARE_FAILURE);
    }
    break;
}

// delete the i/p buffer
PDUM_eAPduFreeAPduInstance(pZPSevent->uEvent.sApsDataIndEvent.hAPduInst);
}
```

在函数 eCLD_OnOffCommandHandler 的处理过程中，根据收到的 On,Off, Toggle 命令，分别调用对应的处理函数。

eCLD_OnOffCommandHandler

→eCLD_OnOffHandleOnCommand

→eCLD_OnOffHandleOffCommand

→eCLD_OnOffHandleToggleCommand

```
PUBLIC    teZCL_Status eCLD_OnOffCommandHandler(
                        ZPS_tsAfEvent                 *pZPSevent,
                        tsZCL_EndPointDefinition      *psEndPointDefinition,
                        tsZCL_ClusterInstance         *psClusterInstance)
```

```
{

    ..............................

    // SERVER
    switch(sZCL_HeaderParams.u8CommandIdentifier)
    {

    case(E_CLD_ONOFF_CMD_ON):
        eCLD_OnOffHandleOnCommand(pZPSevent, psEndPointDefinition, psClusterInstance,
                                                    sZCL_HeaderParams.u8CommandIdentifier);
        break;

    case(E_CLD_ONOFF_CMD_OFF):
        eCLD_OnOffHandleOffCommand(pZPSevent, psEndPointDefinition, psClusterInstance,
                                                    sZCL_HeaderParams.u8CommandIdentifier);
        break;

    case(E_CLD_ONOFF_CMD_TOGGLE):
        eCLD_OnOffHandleToggleCommand(pZPSevent, psEndPointDefinition, psClusterInstance,
                                                    sZCL_HeaderParams.u8CommandIdentifier);
        break;
    ..............................
    }

    /* Generate a custom command event */
    eZCL_SetCustomCallBackEvent(&sOnOffCustomCallBackEvent, pZPSevent,
                        sZCL_HeaderParams.u8TransactionSequenceNumber, psEndPointDefinition->u8EndPointNumber);
    sOnOffCustomCallBackEvent.eEventType = E_ZCL_CBET_CLUSTER_CUSTOM;
    sOnOffCustomCallBackEvent.uMessage.sClusterCustomMessage.u16ClusterId=
                        psClusterInstance->psClusterDefinition->u16ClusterEnum;
    sOnOffCustomCallBackEvent.uMessage.sClusterCustomMessage.pvCustomData = (void *)&sOnOffCallBackMessage;
    sOnOffCustomCallBackEvent.psClusterInstance = psClusterInstance;

    /* Fill in message */
    sOnOffCallBackMessage.u8CommandId = sZCL_HeaderParams.u8CommandIdentifier;

    // call callback
    psEndPointDefinition->pCallBackFunctions(&sOnOffCustomCallBackEvent);

    /* Generate a cluster update event */
    sOnOffCustomCallBackEvent.eEventType = E_ZCL_CBET_CLUSTER_UPDATE;
    psEndPointDefinition->pCallBackFunctions(&sOnOffCustomCallBackEvent);
    ..................................
    return(E_ZCL_SUCCESS);
}
```

当我们考察 eCLD_OnOffHandleOnCommand 函数的具体处理过程时会发现在这个函数中，数据结构 psSharedStruct->bOnOff = 0x01 被修改，从而在逻辑上实现 On 的动作。物理状态改变则在后面的代码中。

```
PRIVATE    teZCL_Status eCLD_OnOffHandleOnCommand(
                            ZPS_tsAfEvent                 *pZPSevent,
                            tsZCL_EndPointDefinition      *psEndPointDefinition,
                            tsZCL_ClusterInstance         *psClusterInstance,
                            uint8                          u8CommandIdentifier)
{
    ……
#if (defined CLD_LEVEL_CONTROL) && (defined LEVEL_CONTROL_SERVER)
    if(eCLD_LevelControlClusterIsPresent(psEndPointDefinition->u8EndPointNumber) == E_ZCL_SUCCESS)
    {
        /* If not already on, set it on */
        if((bool_t)psSharedStruct->bOnOff != TRUE)
        {
            DBG_vPrintf(TRACE_ONOFF, "LC Set to 1");
            eCLD_LevelControlSetOnOffState(psEndPointDefinition->u8EndPointNumber,
                                           TRUE,
                                           CLD_ONOFF_OFF_WITH_EFFECT_NONE);
        }
    }
    else
    {
        psSharedStruct->bOnOff = 0x01;
    }
#else
    psSharedStruct->bOnOff = 0x01;
#endif

    return eStatus;
}
```

函数 eCLD_OnOffHandleOnCommand 修改了 psSharedStruct->bOnOff 数据结构的状态值，改变了灯的逻辑状态。而真正改变 Light 灯的物理状态则在 Endpoint 的注册回调 APP_ZCL_cbEndpointCallback 函数。当改变数据结构状态值后，将会调用 Endpoint 的注册回调函数，事件类型分别是 E_ZCL_CBET_CLUSTER_CUSTOM 和 E_ZCL_CBET_CLUSTER_UPDATE。在这二个事件的处理过程中，将会调用灯的外设驱动程序 vWhiteLightSetLevels 函数，改变灯的物理状态。

```
PRIVATE void APP_ZCL_cbEndpointCallback(tsZCL_CallBackEvent *psEvent)
{
    ……
    switch (psEvent->eEventType)
    {
    ……
    case E_ZCL_CBET_CLUSTER_CUSTOM:
        ……
    case E_ZCL_CBET_CLUSTER_UPDATE:
        if (psEvent->psClusterInstance->psClusterDefinition->u16ClusterEnum == GENERAL_CLUSTER_ID_SCENES)
        {

        }
```

```
        else if (psEvent->psClusterInstance->psClusterDefinition->u16ClusterEnum == GENERAL_CLUSTER_ID_IDENTIFY)
        {
            APP_vHandleIdentify(sLight.sIdentifyServerCluster.u16IdentifyTime);
            if(sLight.sIdentifyServerCluster.u16IdentifyTime == 0)
            {
                tsBDB_ZCLEvent          sBDBZCLEvent;
                /* provide callback to BDB handler for; identify on Target */
                sBDBZCLEvent.eType = BDB_E_ZCL_EVENT_IDENTIFY;
                sBDBZCLEvent.psCallBackEvent = psEvent;
                BDB_vZclEventHandler(&sBDBZCLEvent);
            }
        }
        else
        {
            if (sLight.sIdentifyServerCluster.u16IdentifyTime == 0)
            {
               bUpdateBulb = TRUE;
            }
        }
        break;

    default:
        DBG_vPrintf(TRACE_ZCL, "\nEP EVT: Invalid evt type 0x%x", (uint8)psEvent->eEventType);
        break;

    }

    if (bUpdateBulb)
    {
        vUpdateBulbFromZCL(FALSE);
    }
}

PRIVATE void vUpdateBulbFromZCL(bool_t bResetInterpolation)
{
    ..................
#if (defined CLD_COLOUR_CONTROL) && !(defined DR1221) && !(defined DR1221_Dimic)
    ..................

#elif (defined CLD_COLOUR_CONTROL) && ((defined DR1221) || (defined DR1221_Dimic))
    /* controllable colour temperature tunable white (CCT TW) bulbs */
    ..................

#elif ( defined MONO_WITH_LEVEL)
    ..................
#elif (defined MONO_ON_OFF)
    /*
     * mono on off bulb
     */
```

```
    DBG_vPrintf(TRACE_PATH, "\nJP on_off only bulb");
    vSetBulbState( sLight.sOnOffServerCluster.bOnOff);
#endif
    u8StateChangeTick = BULB_SAVE_DELAY_SEC;
}
```

从 JN-AN-1218-Zigbee-3-0-Light-Bulb 处理流程我们可以分析得知，NXP ZigBee 协议栈已经实现了绝大部分处理代码。用户只需要在端点的注册回调函数 APP_ZCL_cbEndpointCallback 中修改并实现业务功能即可。整个 ZigBee 协议栈涉及的代码比较多，但真正需要用户修改、实现的用户代码其实并不多。往往只需要增加几百行代码即可完成一款新产品开发工作。