如何在 STM32WB 系列上使用 ZigBee®群集模板

引言

本文档介绍了如何在 STM32WB 系列上使用 Zigbee®群集。

Zigbee 应用通常构建在 Zigbee 群集库之上。

本应用笔记说明了此类群集的使用及控制方法。

本文档部分内容受版权 © 2019-2020 Exegin Technologies Limited.保护。经许可转载。

1 概述

本文档适用于 STM32WB 系列基于双核 Arm®的微控制器。

注意： *Arm 是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

1.1 参考文档

- [R1] 05-2374-22《ZigBee PRO 规范第 22 版》
- [R2] 07-5123-07《Zigbee 群集库第 7 版》
- [R3] AN5500《ZSDK API 在 STM32WB 系列上实现 Zigbee®》
- [R4] 07-5356-21《Zigbee 智能能源标准第 1.4 版》

1.2 快速开始

本文档可用于指导开发人员了解使用 Exegin ZCL 群集模板构建应用的要点。每个 Exegin ZCL 群集模板均提供了起始源代码，以便实现完整的群集。该模板提供了 Exegin 规范[R2]、[R4]中定义的强制性 ZCL 命令和属性。

许多命令和属性已成功通过 Zigbee®认证测试。应用的构建包括向特定硬件添加接口，或添加类似应用特定的详细信息。这对于以应用为中心的群集尤为如此，以应用为中心的群集的主要目的是提供应用特定的功能，诸如“开关”群集，对于灯而言，该群集会打开和关闭灯。

除了特定于应用程序的群集之外，还为其他类型的群集（如支持群集和专业群集）提供了模板。基本群集通过提供设备的品牌和型号等信息来支持常规操作。组合，可以将其它群集进行组合管理。第三种类型的群集（即专业群集）独立性较高，并且涉及到应用的部分最少。这些群集提供了一些特殊函数，诸如 Touchlink、CBKE（智能能源安全）和绿色电源。

本文档旨在与定义群集的[R2] ZCL 7 规范和[R3]结合使用，用于 Exegin ZSDK API。

本文档基于[R2]和[R3]构建，为用户使用群集模板构建应用提供了相关知识。

第 2 节“使用群集模板工作”包含有关 API 工作原理的基本信息，和第 3 节“以应用为中心的群集”包含有关群集模板的具体信息。

第 4 节“支持群集”包含构建应用所依赖的基本信息。

第 5 节“特殊专用端点群集”包含有关更专用群集的信息。

注意： *要构建应用，建议从以下清单开始，然后查看后续章节。*

1.3 清单

- 确保参考部分中的文档可用
- 评估应用需要哪些 ZCL 群集
- 参见第 3.1 节“以应用为中心的客户端群集”来实现群集的客户端
- 参见第 3.2 节“以应用为中心的服务器群集”来实现群集的服务器端
- 参见第 4 节“支持群集”用于基本群集、报警群集、群组群集、场景群集和识别群集
- 参见第 5 节“特殊专用端点群集”用于 CBKE 群集、touchlink 群集和绿色电源群集”
- 需要多个群集实例的应用需要多个端点

应用负责：

- 对端点和群集进行实例化
- 对于客户端群集：
  - 根据应用需要将命令和属性请求从本地群集发送到远程服务器群集
  - 在极少数情况下，客户端群集可支持其自有属性
- 对于服务器群集：
  - 在值改变时编写本地属性，或通过回调的形式提供需要的数据
  - 提供回调函数来处理远程命令请求并提供命令响应
  - 与硬件接合

大多数应用还有其他职责，诸如支持用户接口、控制硬件或其他非 Zigbee 相关接口或服务。所有这些职责均超出本文档的范围。

1.4 回调和应用结构

群集模板函数利用应用提供的回调，这些回调具有以下原型：

```
enum ZclStatusCodeT ZbFunction(...,
void (*callback)(...,
void * arg), void *arg);
```

当应用调用此函数时，会提供回调函数和参数。该函数快速返回，当操作完成时，将调用回调。当调用回调时，将传回（回调）在调用原始函数（ZbFunction）时提供的相同参数。

回调通常需要访问 ZSDK 协议栈指针和其他应用数据。该应用还需要在某个位置存储指向群集的指针。通用约定定义了用于存储该信息的应用存储结构。

```
struct 应用；
{
struct ZigbeeT * zb;
struct ZbZclClusterT * a_cluster;
/* other application data */
}
struct application app;
...
status = ZbFunction(..., callbk, &app)
...
void callbk(..., void *arg)
{
struct application *app = (struct application *)arg;
app ->zb ...
app->a_cluster ...
}
```

然而，最好的做法是只披露回调中真正需要的内容。在一些情况下，优先传递群集指针。这也使回调更通用，因为同一回调可与不同的群集实例一起使用。

1.5 群集和端点

协议栈负责建立和维护某个节点与 Zigbee 网络上的其他节点之间的通信（参见[R1]和[R3]）。应用组件建立在这些较低层上并交换 Zigbee 群集库（ZCL）消息。

本文档介绍了如何使用 ZSDK 提供的群集模板构建应用。

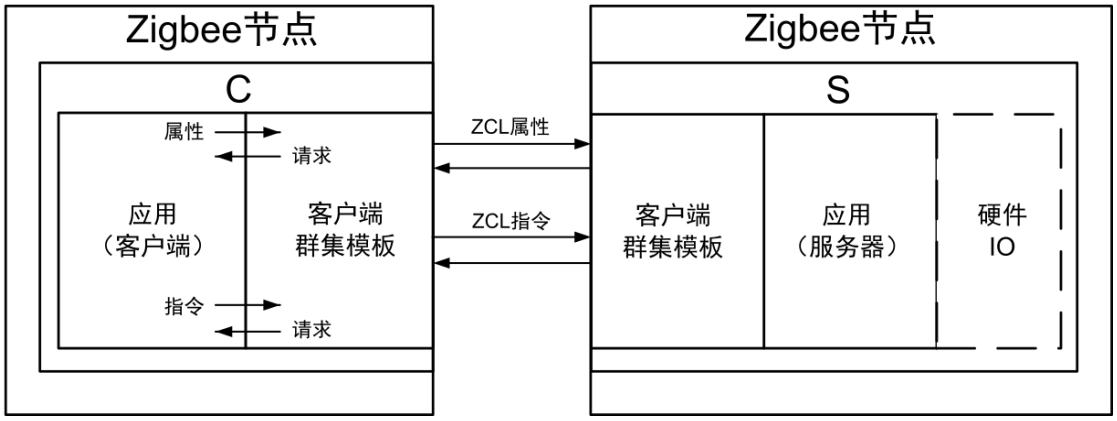
ZCL 由多套有组织的相关功能（即群集）组成。通常，该功能与特定的硬件原件相关联，诸如单个灯或控制灯的开关。与给定设备相关联的所有函数都位于单个端点上。

节点上的每个端点都具有一个唯一的端点 ID（通常称为“端点”），取值范围从 1 到 239。保留端点 0 和 240 到 255 用于特殊用途。

1.6 客户端服务器的关系

群集功能分为服务器端和客户端，服务器端通过一个节点上的端点提供服务，客户端通过另一个节点上的另一个端点访问该服务）。

图 1. 客户端-服务器架构



一个节点通过支持同一群集（举例来说，“开关”群集）的多个实例，可以具有多台物理设备（例如，该节点支持多个灯或多个开关）。因此，每个实例位于唯一的端点上，一个节点上的特定灯（“开关”服务器）通过使用相应的节点端点，与另一个节点上的特定开关（“开关”客户端）相关联。

例如：开关 3 可以位于开关节点上的端点 3 上，并且配置为与灯节点的端点 2 上的灯 2 通信。

给定端点上只允许有一个群集实例，但每个端点通常具有多个群集（如“开关”、基本、报警、场景等）。

应用使用 ZbZclAddEndpoint()函数创建一个或多个端点，该函数已在 Zigbee 群集库头文件 zcl.h 中予以声明。

注意： *也可以使用 APS 函数创建端点。*

对于 ZCL 应用，首选上述 API。

创建新端点时，应用应在 1 和 239 之间选择新的唯一端点，并提供配置文件 ID 和设备 ID。

在以下样例中，我们选择端点 1。配置文件 ID 和设备 ID 的值以 ZCL_PROFILE_和 ZCL_DEVICE_开头，并在头文件 zcl.h 中定义。

```
ZbApsmeAddEndpointReqT add_ep_req;
ZbApsmeAddEndpointConfT add_ep_conf;
memset(&add_ep_req, 0, sizeof(add_ep_req));
memset(&add_ep_conf, 0, sizeof(add_ep_conf));
add_ep_req.endpoint = 1;
add_ep_req.profileId = ZCL_PROFILE_HOME_AUTOMATION;
add_ep_req.deviceId = ZCL_DEVICE_ONOFF_SWITCH;
ZbZclAddEndpoint(app->zb, &add_ep_req, &add_ep_conf);
```

创建端点后，使用 ZbZclClusterSetProfileId()等 ZCL 帮助程序函数来帮助更改端点设置。

注意： *始终建议在使用之前采用 memset 将数据结构归零，以防因内存未初始化而引起的不可预测的行为。*

大多数协议栈函数采用协议栈指针（(struct ZigbeeT 指针作为其第一个参数。大多数 ZCL 函数将采用 ZbZclClusterT 作为其第一个参数（有些则采用 ZigbeeT）。

ZbZclAddEndpoint()的原型通常在协议栈之后。

添加端点请求和确认在头文件 Zigbee.aps.h 中定义。

1.7 端点和简单描述符

当用户创建端点时，协议栈会在内部为每个创建的端点创建一个简单的描述符（参见[R1]第 2.3.2.5 节）。简单描述符具有两个群集列表：“输入”和“输出”。ZCL 服务器群集位于输入列表中，而客户端群集位于输出列表中。

可以使用 ZDO Simple_Desc_req 查询远程设备的简单描述符。然而，实际上，Match_Desc_req ZDO 命令更常用于在远程节点上查找群集。当使用 Match_Desc_req 和 Simple_Desc_req 时，首先使用 Active_EP_req 发现远程节点的端点可能很有用。

有关这些 ZDO 命令的详细信息，请参见[R1]和[R3]。

1.8 群集类型

创建端点后，应用可以创建应用群集并将其添加到该端点。在默认情况下，每个端点都包含基本群集，本文档稍后将对此进行更详细的讨论。

除了基本群集之外，还有其他几个特殊群集，包括：群组群集、场景群集、touchlink 群集、CBKE 群集、报警群集和绿色电源群集。

1.9 以应用程序为中心、支持和特殊群集

Zigbee 应用使用群集构建。大多数群集提供应用功能，诸如：开关、色彩控制、温度测量、智能能量计量等。Zigbee 节点提供端点，每个端点都包含一个或多个以应用为中心的群集的实例。

此外，还有支持群集，诸如群组群集、场景群集和报警群集，这些群集不提供此类应用功能。相反，这些支持群集披露出支持其他群集的功能，使能分别能够对其他群集进行群组寻址、激活其他群集的设置以及记录其他群集生成的报警。

此外，还有一些特殊群集，这些群集由协议栈自动创建，用于提供与应用没有直接关系的特殊功能。这些群集（诸如 CBKE、touchlink 和绿色电源代理基本群集）也通常位于其自己的端点上。

以下章节首先介绍了以应用为中心的群集，然后是支持群集，之后是特殊群集。

1.10 命令：特定于群集与配置文件范围

ZCL 群集提供一组属性和命令（某些群集仅提供属性）。命令分为两种类型：群集特定的命令（特定于该群集，例如基本或 OnOff）和配置文件范围的命令（在每个群集上可用）。配置文件范围的命令的主要用途是访问群集的属性。每个群集都有其自己独特的群集特定的命令。属性主要通过配置文件范围的命令（读取属性、写入属性）进行访问，但配置文件范围的命令还包括其他常规函数，诸如默认响应、配置报告、发现属性和命令等。配置文件范围的命令对于所有群集都是相同的。配置文件范围的命令在 ZCL 7 [R2]的第 2.5 节“通用命令帧”中进行描述。

1.11 寻址和绑定

应用可以使用多种机制来处理从源（通常是客户端）群集到目标（通常是服务器）群集的属性或命令请求。在 Zigbee 网络中使用网络短地址的成本很高，因此自然采用使用短地址的单播寻址。另外也可以使用扩展地址，但这并不常见。当使用群组时，可以使用群组寻址来通过单一请求对多个设备进行多播（参见第 4.2 节“群组群集”）。然而，对于实际应用，大多数 Zigbee 应用采用绑定。

绑定是预配置的源到目标的地址。它们可以通过应用在节点上进行本地配置，也可以使用无线 ZDO 消息，例如通过单独的调试工具进行远程配置。对于特定群集（群集 ID），绑定将本地端点与远程网络短地址和端点相关联。应用通过选择源群集并使用 ZB_APSDE_ADDRMODE_NOTPRESENT 作为目标来使用绑定。然后，APS 层自动扫描绑定表，寻找具有匹配的源端点和群集 ID 的绑定。接着，将请求发送到匹配的所有绑定。无论绑定数如何，只会向应用返回一个响应。

绑定的使用使得应用程序更简单，因为它消除了在每个发送消息的位置确定适当地址的负担，并将其放置在单个中央委托步骤中。应用还可以使用标准 ZDO 服务（诸如 ZbZdoMatchDescReq() 和 ZbZdoSimpleDescReq()）来发现远程端点上的群集，并且使用 ZbApsmeBindReq()创建其自己的本地绑定，或者在某些情况下，使用 ZbZdoBindReq()将绑定发送到远程设备（通常发送回本地节点）。

struct ZbApsAddrT 在寻址中广泛使用。不必在每次寻址绑定时都声明 ZbApsAddrT，而是可以指定帮助程序 ZbApsAddrBinding 以便使用可用的绑定。

对于客户端应用，绑定通常用于处理命令和属性请求。在服务器上，可以配置属性值更改报告。在服务器群集上，由绑定确定报告目标。

需要接收报告的客户端使用 ZbZdoBindReq() 在服务器上创建返回到其自身的绑定，然后使用 ZbZclAttrReportConfigReq()创建“报告配置”来配置报告。然后，服务器在需要发送报告时，使用客户端创建的本地服务器端绑定来了解将报告发送到何处。

1.12 查找和绑定

BDB 提供了一种创建绑定的半自动方法，该方法基于识别群集标识模式构建。查找和绑定过程可以在加入后自动启动（例如，在使用 ZbStartTypeJoin 调用 ZbStartup()时）。在 ZbStartupTbdbComissioningMode 中启用 BDB_COMMISSION_MODE_FIND_BIND 时，进行自动查找和绑定。应用还可以通过调用 ZbStartupFindBindStart()，随时手动启动查找和绑定。

查找和绑定依赖于支持标识服务器群集的目标。在发起设备执行查找和绑定之前，一个或多个目标必须进入识别模式。这些目标通过某些本地操作（诸如按下按钮），或通过来自发起设备或网络上一些其他设备（诸如配网工具）的识别命令，进入识别模式。

查找和绑定发起设备发送广播 IdentifyQuery 命令（zcl_identify_query_request()）。识别模式下的目标以非零识别时间进行响应。然后，发起设备在识别模式下从每个节点请求“简单描述符”（ZbZdoSimpleDescReq()）。接着，发起设备为每个具有匹配远程服务器群集的本地客户端群集，创建本地绑定。务必注意，该过程依赖于在查找和绑定之前在发起设备上实例化的客户端群集。

查找和绑定过程受到多个限制。只能将本地客户端群集绑定到远程服务器群集。通常这是所期望的行为；然而，在某些情况下（诸如智能能源消息传递群集），需要与客户端绑定相反的服务器。

而且，查找和绑定会不加区别地将发起设备客户端端点与目标端点绑定。通常建议仅设置一个客户端端点和一个服务器端点。但在存在多个客户端端点和/或多个服务器端点的应用中，这并非理想的方法。

查找和绑定无法解决所有应用配网需求。应用始终可以通过以下方式建立其自己的绑定：重复使用与查找相同的机制（如 ZDO 命令），并以不同的应用特定方式使用这些机制进行绑定，从而创建所需的绑定。

2 使用群集模板工作

2.1 群集指针-ZbZclClusterT

所有群集都由相同的 ZbZclClusterT 数据类型表示，该数据类型表示特定群集的实例。一般来说，该数据类型的内部结构对应用并无用处；然而却非常重要，因为无论群集的类型如何，该结构都会在整个群集 API 中用作任何群集的通用表示形式。由此，许多 API 都是通用的并且适用于任何群集。例如，ZbZclReadReq()允许读取任何群集中的属性，而 ZbZclDoorLockClientLockReq()只适用于“门锁”客户端群集句柄，但两者都使用相同的 ZbZclClusterT 数据类型。一般来说，区别在于上下文。

2.2 群集命名约定

命名约定要求群集特定的 API 包含群集的名称（在 ZbZclReadReq()和 ZbZclDoorLockClientLockReq()的情况下，没有用于“门锁”客户端群集的群集）。定义用头文件也是用于指示说明，ZbZclReadReq()在一般的“zcl.h”中进行定义，而 ZbZclDoorLockClientLockReq()在群集特定的头文件“zcl.doorlock.h”中定义。最后，如果提供了错误类型的群集句柄，则群集特定的 API 将返回 ZCL_STATUS_FAILURE。重要的一点是，务必检查状态代码以防发生意外行为。

2.3 创建群集的新实例

```
#include "zcl.x.h"
app->x_client_cluster = ZbZclXClientAlloc(zb, endpoint, …)
或
app->x_server_cluster = ZbZclXServerAlloc(zb, endpoint, …)
if (app->x_server_cluster == NULL) {
/* handle error */
```

分配函数在出现错误时返回 NULL，否则返回群集句柄。一般来说，应用从不对该结构的内容进行检查。相反，该句柄在大多数群集库应用中使用。

像大多数 ZSDK API 函数一样，分配函数采用 structZigbeeT *协议栈指针作为其第一个参数。这会将新的群集实例与协议栈实例绑定。从这一点开始，ZCL API 函数采用分配函数返回的 structZbZclClusterT *，即对新创建的群集实例的引用。

分配函数的第二个参数是端点 ID。这会将新创建的群集与该端点绑定。只要任何给定群集只有一个实例，就可以将多个群集实例与同一个端点绑定。

在端点之后，其余参数特定于群集。

具有多个命令的服务器群集通常采用具有多个回调的结构，应用支持的每个命令对应一个回调。应用为其支持的每个命令提供回调。如果应用为整个结构指针或特定命令回调提供 NULL，则群集将以“默认响应”ZCL_STATUS_UNSUPP_CLUSTER_COMMAND 对特定命令（或者如果整个结构指针为 NULL，则针对每个命令）进行响应。

下面是应用如何实现此类回调的样例，从而提供在每当收到 ZCL_ELEC_MEAS_CLI_GET_PROFILE_INFO 命令时所调用的函数 app_get_profile_info()。

```
enum ZclStatusCodeT app_get_profile_info (struct ZbZclClusterT *cluster,
structZbZclAddrInfoT *src_info, void *arg)
{
/* TODO handle get profile info command */
return ZCL_STATUS_SUCCESS;
}
…
struct zcl_elec_meas_callbacks_t callbacks = {
app_get_profile_info,
NULL,
};
…
cluster = ZbZclElecMeasServerAlloc (zb, endpoint, callbacks, app);
…
```

注意： *在该样例中，“获取测量配置文件”回调被声明为 NULL。收到该命令时，自动发送“默认响应”ZCL_STATUS_UNSUPP_CLUSTER_COMMAND。*

enum ZclStatusCodeT 定义了可用的状态代码。

2.4 远程群集与本地群集

使用群集时，有时会产生混淆的一个方面是区分群集是本地的还是远程的。这种混淆部分来自客户端群集和服务器群集之间的区别。服务器群集位于提供功能的网络节点上，并且该服务器功能可通过其本地客户端群集，从远程网络节点进行访问。考虑位于服务器群集上的属性。以下样例展示了远程客户端节点是如何发送写入属性请求的。

2.5 远程写入属性

```
#include “zcl.h”
ZbZclWriteReqT *req;
/* populate req */
memset(&req, 0, sizeof(req));
req.dst.mode = ZB_APSDE_ADDRMODE_NOTPRESENT; /* send to binding */
req.count = 1;
req.attr[0].attrId = ZCL_BASIC_ATTR_ENVIRONMENT;
req.attr[0].type = ZCL_DATATYPE_ENUMERATION_8BIT;
req.attr[0].value = ZCL_BASIC_ENVIRONMENT_MIRROR_SUPPORT;
req.attr[0].length = 1;
status = ZbZclWriteReq (app->cluster, req, read_rsp_callback, &arg);
if (status != ZCL_STATUS_SUCCESS) {
/* handle error */
```

注意： *客户端使用 ZbZclWriteReq() 访问远程属性。一般来说，OTA 消息的名称中包含请求（或 Req）。*

然而，该属性是服务器的本地属性，服务器还必须能够向其写入，以更新将由远程节点读取的值。

2.6 本地写入属性

```
#include “zcl.h”
status = ZbZclAttrIntegerWrite(app->server_cluster, ZCL_BASIC_ATTR_ENVIRONMENT, 0x03);
/* ... */
```

如 Zigbee 规范中所定义的，某些属性为读写，而其他属性为只读。只读意味着无法通过 OTA 写入，例如远程写入。它们必须可由应用在本地写入。需要注意的重要一点是，在服务器上，本地应用始终可以对属性进行写入，甚至对 OTA 只读的属性亦是如此。

2.7 默认 PICS 设置

当群集经 Zigbee 联盟授权的测试机构认证时，每个经认证的群集都需要 PICS（协议实现一致性声明）。

PICS 包括已实现特征清单，包括属性和命令。所提供的群集模板包括交付的群集的 PICS 设置。

以下代码是场景群集中的一个样例：

```
/* PICS.ZCL.Scenes
 *
 * S.S | 真
 * S.C | 真
 *
 * 服务器属性
 * S.S.A0000 | 真
 * S.S.A0001 | 真
 * S.S.A0002 | 真
…
 * 已接收的指令
 * S.S.C00.Rsp | 真
…
```

PICS 注释仅适用于已发布 PICS 文档（诸如 ZCL 7）的群集，并且可用作功能认证的起点。然而，由于特定实现可能会选择向群集添加可选的命令和属性，这需要通过更改单个项目的状态来进行更新。

2.8 向 ZTT 输入 PICS 信息

当使用 ZTT（Zigbee 测试工具）运行测试用例时，必须将群集的头文件中的 PICS 注释中包含的信息手动输入到 ZTT 中。

此条目通过以下方式完成：

1. 打开 ZTT
2. 导航到 PICS 选项卡
3. 向下滚动以查找相应的群集
4. 选择已实现的特征（在 PICS 注释中列为 TRUE 的特征）

ZTT 显示所支持的测试用例并允许用户运行这些测试用例。

3 以应用为中心的群集

ZCL 包括许多应用特定的群集，反映了可能支持的设备的多样性。应用特定的群集定义了设备及其功能。例如，恒温器上的恒温器群集、百叶窗上的窗口覆盖群集或智能能量计上的仪表群集。其他应用特定的群集与特定设备上可能存在的特定功能相关。高级恒温器可支持恒温器用户界面配置群集，而简单恒温器可能不支持。又如，HVAC 控制器可以支持泵的配置和控制、液位控制和“开关”群集，以便为其功能提供支持。

应用特定群集中的共同元素是其支持设备特定的用例。例如，作为恒温器、HVAC 控制器、智能能源控制器或百叶窗的操作。在大多数情况下，应用特定群集提供硬件接口。相比之下，支持群集（诸如基本群集、群组群集、场景群集、报警群集和识别群集）提供一般功能，以便为其他群集或特殊群集（诸如 Touchlink、CBKE 和绿色电源）提供支持，从而提供与设备的主要用途无关的功能。

应用特定群集作为模板交付，这些模板必须通过应用完成。“开关”群集模板处理属性和命令消息处理的详细信息，但应用必须完成诸如“将打开命令转换为设置 GPIO”的详细信息。

所有应用特定群集都遵循通用模式。本节将展示该模式以及如何在应用代码中使用该模式。

表 1. STM32WB Zigbee 群集

群集	头文件	群集	头文件
功率配置	zcl.power.config.h	设备温度配置	zcl.device.temp.h
OnOff	zcl.onoff.h	开/关开关配置	zcl.onoff.swconfig.h
水平控制	zcl.level.h	时间	zcl.time.h
RSSI 位置	zcl.rssi.loc.h	调试群集	Zcl.commission.h
OTA 更新群集	zcl.ota.h	电源配置文件	Zcl.power.profile.h
轮询控制	zcl.poll.control.h	最近网关	zcl.nearest.gw.h
门锁	zcl.doorlock.h	窗口覆盖	zcl.window.h
泵配置和控制	zcl.pump.h	恒温器	zcl.therm.h
风扇控制	zcl.fan.h	除湿控制	zcl.dehum.ctrl.h
恒温器 UI 配置	zcl.therm.ui.h	颜色控制	zcl.color.h
镇流器配置	zcl.ballast.config.h	照度测量	zcl.illum.meas.h
照度水平感测	zcl.illum.level.h	温度测量	zcl.temp.meas.h
压力测量	zcl.press.meas.h	诊断	zcl.diagnostics.h
占用感测	zcl.occupancy.h	IAS 区	zcl.ias_zone.h
IAS 辅助控制设备	zcl.ias_ace.h	IAS 报警设备	zcl.ias_wd.h
Zigbee 语音	zcl.voice.h	仪表识别	zcl.meter.id.h
电气测量	zcl.elec.meas.h	-	-

注意：　在编译该列表之后，可能已添加其他群集。

3.1 以应用为中心的客户端群集

第 2.3 节“创建群集的新实例”涵盖了创建客户端群集实例。应用的作用是保存群集指针（建议的方法是使用可在回调函数中作为“arg”传递的应用存储结构）。然后，每当应用想要使用本地客户端群集访问远程节点上的服务器时，都会使用指向群集实例的指针。每个以应用为中心的客户端群集均提供一个通用 API。本节介绍了该通用 API。对于属性名称或命令函数等群集具体细节，请参阅具体群集的头文件（例如，对于“开关”群集，请参阅 zcl.onoff.h）。ZCL 群集头文件同时包括客户端和服务器 API。属性和命令在 ZCL 7 [R2]中定义，但根据 ZSE 规范 [R4]定义的智能能源群集除外。

3.1.1 从客户端群集访问远程服务器属性

远程服务器上的属性通过本地客户端群集进行访问。

注意： *在某些情况下，客户端群集上存在从服务器群集访问的属性。对于大多数群集，唯一的客户端属性是每个客户端和服务器群集上都必须存在的群集版本 ZCL_GLOBAL_ATTR_CLUSTER_REV。某些客户端群集具有属性，如所有群集特定的 OTA 和 DRLC 群集属性，都位于客户端。计量群集具有几乎所有的服务器属性和少量客户端属性（群集版本除外），而设备管理在服务器端和客户端都具有属性。*

属性可以是 R（只读）、W（只写）或 RW（读写），另外还有一些是 P（可报告）。每个群集头文件都包含一个形式为例如 ZCL_<cluster>_ATTR_<name>的匿名枚举，该匿名枚举定义了该群集的属性 ID。下面的样例为 SensorType 属性的远程值的读取：

```
#include “zcl.occupancy.h”
void handle_read_rsp(const ZbZclReadRspT *rsp, void *arg)
{
struct application *app = (struct application *)arg;
if (rsp->status == ZCL_STATUS_SUCCESS) {
app->remote_occupancy = (uint8_t)*rsp.attr[0].value;
}}…
enum ZclStatusCodeT status;
ZbZclReadReqT req;
memset(&req, 0, sizeof(req));
req.dst = &ZbApsAddrBinding;
req.count = 1;
req.attr[0] = ZCL_OCC_ATTR_SENSORTYPE;
status = ZbZclReadReq(app->occ_client, req, handle_read_rsp, app);
if (status != ZCL_STATUS_SUCCESS) {
/* handle error */
…
```

注意： *目标为“绑定”，其假定已经建立了从客户端到远程服务器的绑定。*

3.1.2 向远程节点发送命令

一般而言，客户端向服务器发送命令请求，但与属性一样，偶尔会出现异常。客户端群集模板提供的函数允许向服务器发送命令请求。这些命令的语法为：

ZbZcl[Cluster][Client/Server]Command[Name]Req()

例如，ZbZclMeterClientCommandGetProfileReq()从仪表群集客户端发送“Get Profile”命令请求。许多命令的有效负载如 ZCL 或 ZSE 规范中所定义的。下表展示了仪表群集“Get Profile”命令（参见[R2]文档）：

表 2. 获取配置文件命令有效负载

八位字节	1	4	1
数据类型	8 位枚举(1)	UTCTime	无符号 8 位整数
字段名称	间隔通道	结束时间	周期数

1. CCB 1077

群集模板为每个命令有效负载提供 C 结构。在这种情况下，ZbZclMeterClientGetProfileReqT 在“zcl.meter.h”中定义。应用对这些结构之一进行声明，并将其传递给命令请求。

以下代码片段说明了将命令从客户端发送到服务器并接收响应的一般过程。

```
struct application app;
...
void get_profile_callback(struct ZbZclCommandRspT * rsp, void *arg)
{
struct application *app = (struct application *)arg;
if (rsp.status == ZCL_STATUS_TIMEOUT) {
/* server did not respond within timeout */
}
else if(((rsp.hdr.frameCtrl.frameType & ZCL_FRAMETYPE_CLUSTER)==0)&&
(hdr.cmdId == ZCL_COMMAND_DEFAULT_RESPONSE)) {
/* server returned a default response, indicating error */
}
else if(rsp.status == ZCL_STATUS_SUCCESS) {
/* a valid response was returned */
}
else {
/* an error occurred */
}}
...
enum ZclStatusCodeT status;
struct ZbZclMeterClientGetProfileReqT req;
memset(&req, 0, sizeof(req));
interval_channel = ZCL_METER_SAMPLE_TYPE_CONSUMP_DELIV;
end_time = 0; /* zero gets the most recent */
number_of_periods = 1;
status = ZbZclMeterClientCommandGetProfileReq(app.cal_cluster, & ZbApsAddrBinding, & req,
get_profile_callback, &app);
```

应用回调必须对以下四种可能的情况进行处理：

1. 按预期收到 ZCL 响应
2. 服务器未在所需时间量内响应。当 NWK 和 APS 确认均未发生或存在有效 ZCL 响应（“默认响应”或“群集命令”特定响应）时，可能会导致此问题
3. 收到指示错误条件的“默认响应”
4. 返回 ZCL 错误状态

3.1.3 客户端如何从远程服务器群集接收报告

如第 1.11 节“寻址和绑定”中所述，客户端配置远程服务器，以便通过使用 ZbZdoBindReq()向远程服务器发送绑定来创建回到自身的绑定，并使用 ZbZclAttrReportConfigReq()在服务器群集中创建报告配置，向该远程服务器发送报告。然而，在服务器上执行该配置之前，客户端应配置本地应用回调以接收报告。例如：

```
void receive_report (struct ZbZclClusterT *cluster, ZbApsdeDataIndT *data,
uint16_t id, enum ZclDataTypeT type, const uint8_t *payload, uint16_t len)
{
/* handle report */
}
...
struct ZbZclClusterT * client_cluster;
ZbZclClusterReportCallbackAttach(client_cluster, receive_report);
```

每当服务器基于报告配置确定需发送报告时，服务器都会使用客户端已在服务器的绑定表中创建的绑定，将报告发送回客户端。当客户端群集收到具有如上所示配置的回调的报告时，将使用收到的报告调用此回调。然后，应用可根据需要对报告进行处理。

3.2 以应用为中心的服务器群集

前文第 3.1 节“以应用为中心的客户端群集”涵盖了与客户端群集的应用需求相关的信息。此外，本节还涵盖服务器端。

服务器群集为使用网络上的其他节点提供服务，并通过本地服务器群集公开这些服务。服务器群集模板，必须结合应用的具体实现，才能实现完整的功能。例如，“开关”服务器群集模板处理 ZCL 消息传递的机制，但如何物理打开或关闭硬件的详细信息都必须在应用中实现。本节介绍了应用如何实现以应用为中心的服务器群集。

与客户端群集类似，应用必须按照第 2.3 节“创建群集的新实例”中所述创建服务器群集，并维护服务器群集指针。然而，应用需要服务器群集指针的频率低于需要客户端群集指针的频率。

3.2.1 向群集添加可选属性

在默认情况下，服务器群集模板仅创建强制性属性。如果应用需要定义其他可选属性，则在创建服务器群集实例后，通过使用新的可选属性调用 ZbZclAttrAppendList()，即可轻松完成。

```
static const struct ZbZclAttrT optional_attr_list[] = {
{
ZCL_METER_SVR_ATTR_CURSUM_RECV, ZCL_DATATYPE_UNSIGNED_48BIT,
ZCL_ATTR_FLAG_NONE, 0, NULL,
{0, 0}, {0, 0}
},
{
ZCL_METER_SVR_ATTR_MAX_DMND_RECV, ZCL_DATATYPE_UNSIGNED_48BIT,
ZCL_ATTR_FLAG_NONE, 0, NULL,
{0, 0}, {0, 0}
},
...
};
...
app->meter_server = ZbZclMeterServerAlloc(zb, endpoint, NULL, &app);
if (ZbZclAttrAppendList(&app->meter_server, optional_attr_list,
ZCL_ATTR_LIST_LEN(optional_attr_list)) != ZCL_STATUS_SUCCESS) {
/* handle error */
}
…
```

除了添加新属性之外，还可以覆盖强制性属性的内置定义。

3.2.2 通过直接写入更新属性值

有两种方法用于更新属性值。在默认情况下，在群集本身内对属性值进行维护。服务器应用可使用一组帮助程序函数更新该本地值，诸如 ZbZclIntegerWrite() 用于数值属性，而 ZbclAttrStringWriteShort()/ZbclAttrStringWriteLong()用于字符串属性。例如，当决定更新上一样例中的属性值时，应用可以调用：

```
long long max_demand_received;
ZbZclAttrIntegerWrite(&app->meter_server, ZCL_METER_SVR_ATTR_MAX_DMND_RECV,
max_demand_received);
```

由于该值存储在群集中，因此在某些情况下，应用可能需要该属性，尤其是当其可能由远程客户端写入时亦是如此。本地属性值可使用帮助程序 ZbZclAttrIntegerRead()随时读取。这些数据类型特定的帮助程序适用于最常见的数据类型。访问任何本地群集服务器属性的通用 API 为 ZbZclAttrRead()和 ZbZclAttrWrite()。这些通用 API 使用起来不太方便，因为属性值通过 OTA（通常为小端序）进行存储。对于具有端序的 ZCL 数据类型，putle()函数（来自 pletoh.h）应当在与 ZbZclAttrWrite()一起使用之前，用于从主机转换为小端序形式；并且 pletoh()函数用于在应用中使用之前，使用 ZbZclAttrRead()将读取值转换为主机端序。

该推送方法是服务器可维护其属性值的一种方法。同时也可以使用下一节所述的回调。

3.2.3 使用群集属性回调

应用可以提供回调函数在访问属性时执行读取（或写入），而无需应用控制属性值。回调尤其适用于属性值由硬件（诸如 GPIO）中物理实体确定的情况。下面是提供读取回调的样例，该读取回调访问硬件中表示的外部内存映射值：

```
static const struct ZbZclAttrT optional_attr_list[] =
{{
ZCL_METER_SVR_ATTR_CURSUM_RECV, ZCL_DATATYPE_UNSIGNED_48BIT,
ZCL_ATTR_FLAG_CB_READ, 0, receive_callback,
{0, 0}, {0, 0}
},
#include "pletoh.h"volatile extern uint64_t current_sum_rcv;
enum ZclStatusCodeT receive_callback(struct ZbZclClusterT *cluster, struct ZbZclAttrCbInfoT
*info)
{switch(info->type)
{case ZCL_ATTR_CB_TYPE_READ:putle48(info->zcl_data, current_sum_rcv);
return ZCL_STATUS_SUCCESS;
default : return ZCL_STATUS_FAILURE;
}
return ZCL_STATUS_SUCCESS;
}
```

回调的回调属性标志值为 eZCL_ATTR_FLAG_CB_READ

ZCL_ATTR_FLAG_CB_WRITE（ZCL_ATTR_FLAG_CB_NOTIFY）是在本地而不是无线写入时的写入特例）。

3.3 实现服务器端命令处理程序

当群集支持命令时（如前所述，一般是服务器端独有），应用负责将回调函数的结构传递给群集实例化“Alloc”函数。

```
static enum ZclStatusCodeT restart_device(struct ZbZclClusterT *clusterPtr,
struct ZbZclCommissionClientRestartDev *req,
structZbZclAddrInfoT *srcInfo, void *arg)
{
if ((req.options & ZCL_COMMISSION_OPTIONS_IMMEDIATE) != 0) {
/* reset immediately */
}
else {
...
return ZCL_STATUS_SUCCESS;
}
struct ZbZclCommissionServerCallbacksT callbacks =
{
restart_device,
NULL,
NULL,
NULL,
}
...
app->commission_server = ZbZclCommissionServerAlloc(zb, profile, aps_secured,
&callbacks, app);
```

在该样例中，服务器应用仅实现重新启动设备 ZCL_COMMISSION_CLI_CMD_RESTART_DEVICE）ZCL 命令。“重新启动设备”命令的有效负载被解压缩，并作为 C 结构被提供给回调。“Save Startup”、“Restore Startup”和“Reset Startup”回调未定义。如果该群集收到这些命令，则会将状态为 ZCL_STATUS_UNSUPP_CLUSTER_COMMAND 的“默认响应”发送回客户端。

如果命令回调返回除 ZCL_STATUS_SUCCESS 以外的任何内容，则发送默认响应，除非客户端已请求在成功的情况下发送“默认响应”。回调可以通过返回 ZCL_STATUS_SUCCESS_NO_DEFAULT_RESPONSE 来覆盖此行为，只有在回调返回 ZCL 响应消息时才能执行此操作。

在以下样例中，服务器请求回调定位了所请求的信息，形成了响应信息（rsp）的结构，并使用服务器群集 ZbZclMeterServerSendGetProfileRsp()，将 ZCL 响应发送回源客户端。

```
static enum ZclStatusCodeT
meter_get_profile(struct ZbZclClusterT *clusterPtr,
void *arg,
struct ZbZclMeterClientGetProfileReqT *req,
struct ZbZclAddrInfoT * srcInfo)
{ZbZclMeterServerGetProfileRspT rsp;
/*
Obtain Profile info matching request */
...
memset(&rsp, 0, sizeof(rsp));
rsp.end_time = req ->end_time;
rsp.status = ZCL_METER_PRFL_STATUS_SUCCESS;
rsp.profile_interval_period = profile[i].profile_interval_period;
rsp.number_of_periods = profile[i].number_of_periods;
rsp.profile_data = profile_data;
rsp.profile_length = ZbZclMeterFormSampledData(profile_data, sizeof(profile_data),samples,
n);
if (rsp.profile_length < 0) {return ZCL_STATUS_INSUFFICIENT_SPACE;
}
ZbZclMeterServerSendGetProfileRsp(clusterPtr, srcInfo, &rsp);
return ZCL_STATUS_SUCCESS;}
```

注意： *它如何返回 ZCL_STATUS_SUCCESS 通知协议栈它已对响应进行处理。错误状态会导致默认响应，其中包含所提供的 ZCL_STATUS_ 错误值。*

在少数情况下，客户端群集支持命令。在这些罕见情况下，所有程序都适用，客户端和服务器的角色只是颠倒过来。

4 支持群集

4.1 基本群集

基本群集的特殊之处在于，服务器实例由协议栈自动创建并添加到每个端点。节点可通过在其端点之一上声明客户端基本群集，访问远程节点的服务器群集。

与其他群集不同，基本服务器群集为一种单例群集，即基本服务器群集只有一个实例并且出现在每个端点上。这是因为，基本群集中的信息（诸如制造商和型号）全局适用于整个节点。为了访问基本群集的功能，提供了一种特殊的帮助程序函数 ZbZclBasicWriteDirect()，参见头文件“Zigbee.h”。该函数使用设备上的任何有效端点以及要更改的属性的详细信息。更新的属性值反映在所有现有和未来端点上的基本群集中。

```
char * mfr_str = “Zigbee”;
uint8_t zb_mfr_str[7];
/* convert C string to Zigbee string */
memcpy(& zb_mfr_str[1], mfr_str, strlen(mfr_str));
zb_mfr_str[0] = strlen(mfr_str);
ZbZclBasicWriteDirect (app-> zb, ep, ZCL_BASIC_ATTR_MFR_NAME, zb_mfr_str, 7);
```

基本群集还支持两种报警：一般硬件故障和一般软件故障。这些报警的生成由“AlarmMask”属性控制。本地应用可通过本地调用 ZbZclBasicPostAlarm()来发布这些报警之一，然而，AlarmMask 属性的相应位可能会阻止其被处理。

基本服务器支持一个命令，即 ZCL_BASIC_RESET_FACTORY。应用必须向协议栈注册回调。在其节点收到其中一个重置为出厂默认值时，调用该回调。收到该回调时，应用负责将所有群集中所有属性的值重置为其出厂默认值。

4.2 群组群集

群组群集提供了一种管理端点的群组管理的机制。通过向端点添加群组群集，可以通过“群组群集”命令对该端点的群组进行远程管理。群组为 APS 协议栈层的一项特征，并且群组存储在群组表内部。应用使用 APS API（诸如 ZbApsmeAddGroupReq()和 ZbApsmeRemoveGroupReq()）访问其本地端点上的群组。群组服务器群集允许远程应用（通过群组客户端群集）以与本地应用使用 APS API 相同的方式，对端点的群组成员身份进行管理。群组群集提供诸如“添加”、“删除”、“全部删除”、“查看”、“获取成员身份”和“如果标识则添加群组”（其链接到 ZCL 识别群集）等命令。有关更多详细信息，请参见第 3.6 节“群组”[R2]和头文件 zcl.groups.h。

4.3 场景群集

场景群集提供了将设置块应用于与“场景服务器”群集位于同一端点上的另一个或多个群集的功能。只有少数选择群集支持场景。在群集规范中，这些群集有一个标题为“场景表扩展”开关的附加部分，有关 “开关”群集请参见第 3.8.2.6 节 [R2]，有关颜色请参见第 5.2.2.5 节，有关“恒温器”请参见第 6.3.2.6 节，并且有关“窗口覆盖”请参见第 7.4.2.4 节。该场景表扩展部分定义了一组要应用于该群集中的属性的特定属性值。例如，“窗口覆盖场景扩展表”由“当前位置提升百分比”和“当前位置倾斜百分比”属性的值按该顺序组成。场景群集将该数据存储在“扩展字段”中（例如图 3-19）。对于场景群集，“扩展字段”是一种不透明的数据块，其应用于对其进行解释和说明的特定群集。

激活场景后，特定的“扩展字段”将应用于相应的特定群集，这些群集根据群集“场景表扩展”部分中的定义对其进行解释。在群集中，这些属性的值从当前值过渡到该群集的已激活场景“扩展字段”中的值。根据属性的不同，该过渡可能不会立即发生，而是可能如场景定义中的“过渡时间”所定义般连续发生。

“场景群集”需要使用群组寻址，因此“群组”群集通常与“场景群集”位于同一端点上。例如，给定端点可以具有“群组”、“场景”、“开关”和“窗口覆盖”等群集（以及“基本”群集）。有关更多信息，请参见第 3.7 节“场景”[R2]和头文件 zcl.scenes.h。

4.4 识别群集

在配网期间，能够了解每个节点非常重要。

识别群集通过引入外部可见的识别模式（如闪烁的 LED、闪烁的显示背光等）来协助此过程，并且可通过无线方式访问。启动后，识别模式在“识别时间”给定的时间间隔内保持活动状态，之后会自动结束。“识别”群集可本身单独使用，但它也是 Touchlink 程序整体所必需的。

请参见第 3.5 节“识别”[R2]和头文件 zcl.identify.h。

4.5 报警群集

报警是在某些群集内出现的条件，这些群集在外部对网络上的其他设备感兴趣。每种报警条件都具有一个关联的报警代码，该代码在该群集中定义，并且通常与一个或多个属性相关联。支持报警和所定义的报警代码的群集要求其端点还具有“报警”群集，这在相应的群集部分中指定。

例如，设备温度群集可以生成两种报警：“设备温度过低”（代码 0x00）和“设备温度过高”（代码 0x01），二者均与设备温度相关。这些报警中的每一个都由两个属性（阈值和驻留时间）控制，用于确定报警条件何时出现。设备温度群集当确定已出现报警条件（低或高）时，会将相应的报警代码发布到其所在的同一端点上的“报警”群集。发起群集在确定报警条件（取决于群集定义）时，会通过调用以下命令发布报警：发起群集内的 ZbZclClusterSendAlarm()（在 zcl.h 中定义）。

报警群集将报警代码和发起群集 ID 记录在可查询的报警日志中。请注意，报警代码只能在发起群集的上下文中进行解释。

报警群集在其“报警日志”中添加条目后，向端点上绑定到“报警群集”的任何客户端发送“报警”命令。这样，客户端只需绑定到“报警”群集即可接收来自该端点上所有群集的报警。

另外，某些群集支持报警掩码。这令外部客户端能够启用和禁用单个报警。请注意，该报警掩码为全局性的，在报警掩码中启用或禁用报警会影响任何和所有客户端。

汇总某些群集支持报警，并针对外部客户端可能感兴趣的条件定义其自己的报警代码。当群集支持报警时，该群集要求“报警”群集也存在于同一端点上，以便记录和发送报警。大多数支持报警的群集还具有报警掩码属性，该报警掩码属性允许一个外部客户端启用或禁止报告特定报警。有关更多信息，请参见[R2]中的第 3.11 节“报警”和头文件 zcl.alarms.h

报警服务器群集在将报警添加到报警日志时，会向报警应用时间戳。该时间戳由必须设置在同一端点上时间群集提供，并且在通过 ZbZclAlarmServerAlloc()调用创建报警服务器群集时，必须向报警服务器群集提供指向时间群集的指针。

5 特殊专用端点群集

某些特殊群集位于专用端点上，这些端点专用于该特定功能。这些群集包括 Touchlink 群集、CBKE 群集和绿色电源群集。对于 touchlink 群集和绿色电源群集，它们位于专用端点上的原因之一在于其使用特殊的配置文件 ID。

选择 BDB_COMMISSION_MODE_TOUCHLINK 时，将创建 touchlink 群集和端点。协议栈创建用户在 ZbStartupT（tl_endpoint 和 bind_endpoint）中设置的特殊 touchlink 端点。

智能能源应用通过在 ZbStartupT 的 security.cbke 子部分中设置 suite_mask，触发协议栈创建 CBKE 群集。当其中一个套件位于 suite_mask （即 ZCL_KEY_SUITE_CBKE_ECMQV 或 ZCL_KEY_SUITE_CBKE_ECMQV）中时。启用后，协议栈创建 CBKE 端点，该端点在默认情况下为 240 但可由应用进行更改并附加 CBKE 群集。

绿色电源（绿色电源代理基本）群集由协议栈在保留端点 242 上自动创建（协议栈的智能能源版本除外）。代理基本群集与应用没有交互。

智能能源和非智能能源应用使用协议栈的不同版本。协议栈构建机制具有许多 make 变量，这些变量导出为相同名称的 C 宏。它们使用前缀 CONFIG_ZB_。对于协议栈的智能能源版本，定义了 CONFIG_ZB_ZCL_CBKE，但未定义 CONFIG_ZB_GREENPOWER、CONFIG_ZB_ZCL_TL_INITIATOR 或 CONFIG_ZB_ZCL_TL_TARGET。如上所述，智能能源版本自动创建 CBKE 群集，但不包括或不支持 touchlink 群集或绿色电源代理基本群集。

协议栈的非智能能源版本未定义 CONFIG_ZB_ZCL_CBKE，因此不包括或不支持 CBKE。它们定义了 CONFIG_ZB_GREENPOWER，从而创建绿色电源代理基本群集和端点。它们还定义了 CONFIG_ZB_ZCL_TL_INITIATOR 或 CONFIG_ZB_ZCL_TL_TARGET，从而支持 touchlink 发起设备或 touchlink 目标功能。智能能源、touchlink 发起设备和 touchlink 目标都需要单独的协议栈版本。

版本历史

表 3. 文档版本历史

日期	版本	变更
2020 年 7 月 24 日	1	初始版本
2020 年 8 月 25 日	2	文档发布范围由限于 ST 公司变为公开发布

目录

表格索引

图片目录

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“意法半导体”）保留随时对 ST 产品和/或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于意法半导体产品的最新信息。意法半导体产品的销售依照订单确认时的相关意法半导体销售条款。

买方自行负责对意法半导体产品的选择和使用，意法半导体概不承担与应用协助或买方产品设计相关的任何责任。

意法半导体不对任何知识产权进行任何明示或默示的授权或许可。

转售的意法半导体产品如有不同于此处提供的信息的规定，将导致意法半导体针对该产品授予的任何保证失效。

ST 和 ST 标志是意法半导体的商标。关于意法半导体商标的其他信息，请访问 www.st.com/trademarks。其他所有产品或服务名称是其各自所有者的财产。本文档中的信息取代本文档所有早期版本中提供的信息。

© 2023 STMicroelectronics - 保留所有权利