POWERLINK 应用层

王谨秋

POWERLINK 开发板，POWERLINK 协议软件和技术支持，IO 模块，运动控制器，工业机器人控制器。

QQ 群： 151181908

联系人： 王先生

手机：13917489045（微信同号）

邮箱：openpowerlink@163.com

网址： http://www.openat.cn

第五章 POWERLINK 应用层

POWERLINK 技术规范规定的应用层为 CANopen，但是 CANopen 并不是必须的，用户可以根据自己的需要自定义应用层，或者根据其他行规编写相应的应用层。

无论是 openPOWERLINK 还是前面提到的 HDL POWERLINK 都可以使用本章介绍的应用层软件。

本章节将介绍 CANopen 应用层。

5.1 CANopen 应用层

POWERLINK 的应用层遵循 CANopen 标准。CANopen 是一个应用层协议。他为应用程序提供了一个统一的接口，使得不同的设备与应用程序之间有统一的访问方式。

CANopen 协议有三个主要部分：PDO，SDO 和对象字典 OD。

PDO ：过程数据对象

可以理解为在通信过程中，需要周期性、实时传输的数据。

SDO ：服务数据对象

可以理解为在通信过程中，非周期性传输、实时性要求不高的数据，例如网络配置命令，偶尔要传输的数据等。

OD：对象字典

可以理解为所有参数/通信对象的集合。

5.2 对象字典-OD

什么是对象字典？对象字典就是很多对象（object）的集合。那么什么又是对象呢？一个对象可以理解为一个参数。假设有一个设备，该设备有很多参数。CANopen 通过给每个参数一个编号来区分参数，这个编号就叫做索引（Index），这个索引用一个 16bits 的数字表示。如果这个参数又包含了很多子参数，那么 CANopen 又会给这些子参数分别分配一个子索引（SubIndex），用一个 8bits 的数字来表示。因此一个索引和一个子索引就能明确的标示出一个参数。

一个参数除了具有索引和子索引信息外，还应该有参数的数据类型（8bits 还是 16bits？有符号还是无符号？），还需要有访问类型（可读的还是可写的，还是可读写的？），还有默认值等等。因此一个参数需要有很多属性来描述他，所以一个参数也就成了一个对象 object，所有对象的集合就构成了对象字典（object dictionary），简称 OD。

在 POWERLINK 对 OD 的定义和声明在 objdict.h 文件中。这个文件采用宏定义的方式，定义了一个数据结构，如图 5-1。

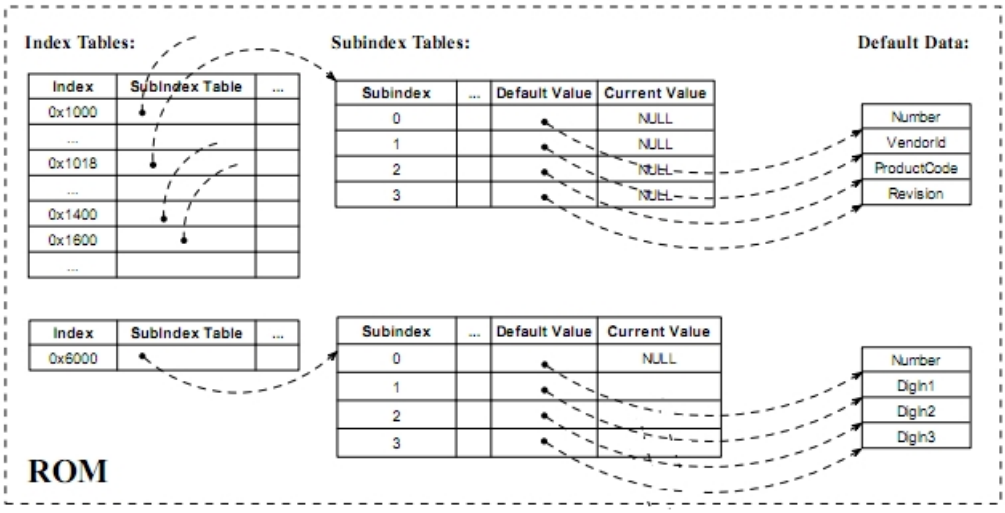


图5-1 CANopen对象字段结构

下面是一个object定义的实例：

EPL_OBD_BEGIN_INDEX_RAM(0x6000, 0x02, NULL)

EPL_OBD_SUBINDEX_RAM_VAR(0x6000, 0x00, kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1)

EPL_OBD_SUBINDEX_RAM_USERDEF(0x6000, 0x01, kEplObdTypUInt8, kEplObdAccVPR, tEplObdUnsigned8, Sendb1, 0x0)

EPL_OBD_END_INDEX(0x6000)

1．索引：16bits的无符号整数，可以把索引看作是一个object的编号，用来对对象字典内的所有object进行寻址。

2．子索引：8 bits的无符号整数，可以把子索引看作是一个object的子编号。索引和子索引组合起来，就唯一确定了一个object。在POWERLINK中，是通过索引和子索引来寻址object的。

3．默认值：object的初始默认值。

4．对象类型：用来描述object的类型，是字符串，是变量，还是时间等，如表5-1所示object的类型说明。

类型	描述
kEplObdTypBool	Boolean (value 0x0001)

kEplObdTypInt8	signed integer 8-bit (value 0x0002)
kEplObdTypInt16	signed integer 16-bit (value 0x0003)
kEplObdTypInt32	signed integer 32-bit (value 0x0004)
kEplObdTypUInt8	unsigned integer 8-bit (value 0x0005)
kEplObdTypUInt16	unsigned integer 16-bit (value 0x0006)
kEplObdTypUInt32	unsigned integer 32-bit (value 0x0007)
kEplObdTypReal32	real 32-bit (value 0x0008)
kEplObdTypVString	visible string (value 0x0009)
kEplObdTypOString	octet string (value 0x000A)
kEplObdTypTimeOfDay	time of day (value 0x000C)
kEplObdTypTimeDiff	time difference (value 0x000D)
kEplObdTypDomain	domain (value 0x000F)
kEplObdTypInt24	signed integer 24-bit (value 0x0010)
kEplObdTypReal64	real 64-bit (value 0x0011)
kEplObdTypInt40	signed integer 40-bit (value 0x0012)
kEplObdTypInt48	signed integer 48-bit (value 0x0013)
kEplObdTypInt56	signed integer 56-bit (value 0x0014)
kEplObdTypInt64	signed integer 64-bit (value 0x0015)
kEplObdTypUInt24	unsigned integer 24-bit (value 0x0016)
kEplObdTypUInt40	unsigned integer 40-bit (value 0x0018)
kEplObdTypUInt48	unsigned integer 48-bit (value 0x0019)
kEplObdTypUInt56	unsigned integer 56-bit (value 0x001A)
kEplObdTypUInt64	unsigned integer 64-bit (value 0x001B)

表5-1 对象数据类型说明

5．数据类型：object的数据类型的 C语言定义。例如该object的对象类型是kEplObdTypUInt8，那么它的数据类型定义应该是 unsigned char 。由于在EplObd.h文件中对各种数据类型做了宏定义。如typedef unsigned char tEplObdUnsigned8; 所以

可以用tEplObdUnsigned8替换unsigned char。

对象类型和数据类型的对应关系如下表5-2所示：

对象类型	数据类型
kEplObdTypBool	tEplObdBoolean
kEplObdTypInt8	tEplObdInteger8
kEplObdTypInt16	tEplObdInteger16
kEplObdTypInt24	tEplObdInteger24
kEplObdTypInt32	tEplObdInteger32
kEplObdTypInt40	tEplObdInteger40
kEplObdTypInt48	tEplObdInteger48
kEplObdTypInt56	tEplObdInteger56
kEplObdTypInt64	tEplObdInteger64
kEplObdTypUInt8	tEplObdUnsigned8,
kEplObdTypUInt16	tEplObdUnsigned16,
kEplObdTypUInt24	tEplObdUnsigned24
kEplObdTypUInt32	tEplObdUnsigned32,
kEplObdTypUInt40	tEplObdUnsigned40
kEplObdTypUInt48	tEplObdUnsigned48
kEplObdTypUInt56	tEplObdUnsigned56
kEplObdTypUInt64	tEplObdUnsigned64
kEplObdTypReal32	tEplObdReal32
kEplObdTypReal64	tEplObdReal64
kEplObdTypTimeOfDay	tEplObdTimeOfDay
kEplObdTypTimeDiff	tEplObdTimeDifference
kEplObdTypVString	tEplObdVString
kEplObdTypOString	tEplObdOString

表5-2 对象类型和数据类型的对应关系

当对象类型为kEplObdTypInt8时，相应的数据类型应该为 tEplObdInteger8。否则，会出现错误。

6. 访问类型：指object是可读的，可写的，还是常量等。各访问权限对应的值和相应的描述如表5-3所示。

访问权限	值	描述
kEplObdAccRead	0x01	该 object 的数值为只读的
kEplObdAccWrite	0x02	该 object 的数值为可写的
kEplObdAccConst	0x04	该 object 的数值为常量
kEplObdAccPdo	0x08	该 object 可以被映射为 PDO，通常和 kEplObdAccVar 一起使用
kEplObdAccRange	0x20	该 object 的数值具有上下限
kEplObdAccVar	0x40	该 object 是一个变量，放置在应用程序中
kEplObdAccStore	0x80	该 object 的数值可以被保存在非易失的存储器中

表5-3 对象的访问类型

一个object 可以具有一个权限，也可以具有多个权限，例如某个object 的访问类型为：可读写的，而且可以被映射为PDO的，那么它的权限值应该为kEplObdAccVar 或kEplObdAccPdo 或kEplObdAccWrite 或kEplObdAccRead。

在EplObd.h文件中，对一些常用的访问类型做了预定义，例如我们刚刚说的可读写，而且可以被映射为PDO的权限类型定义如下：

```
#define kEplObdAccVPRW (kEplObdAccVar |kEplObdAccPdo | kEplObdAccWrite | kEplObdAccRead)
```

在objdict.h中，object的定义实例：

```
EPL_OBD_BEGIN_INDEX_RAM(0x6000, 0x02, NULL)
        EPL_OBD_SUBINDEX_RAM_VAR(0x6000, 0x00, kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1)
        EPL_OBD_SUBINDEX_RAM_USERDEF(0x6000, 0x01, kEplObdTypUInt8,
```

kEplObdAccVPR, tEplObdUnsigned8, Sendb1, 0x0)

EPL_OBD_END_INDEX(0x6000)

EPL_OBD_BEGIN_INDEX_RAM(0x6000, 0x02, NULL)是 object 的头。

EPL_OBD_END_INDEX(0x6000)是 object 的尾。这里的 0x6000 是这个 object 的索引; 0x02 是指它有两个子 object； NULL 是回调函数指针，对于简单应用，这里不用考虑。

接下来的 EPL_OBD_SUBINDEX_RAM_VAR(0x6000, 0x00, kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x1) 是第一个子 object。其中 0x6000 是索引；0x00 是子索引； kEplObdTypUInt8 是对象类型，无符号 8 bits； kEplObdAccConst 是访问类型，这里为常量； tEplObdUnsigned8 是数据类型，无符号 8 bits； number_of_entries 是 object 的名字； 0x1 是默认值，这里之所以默认值为 1，原因是通常子索引为 0x00 的子 object 用来表示有多少个有效的子 object。也就是除了子索引为 0x00 的子 object 外,有多少个有效可用的子 object。这里我们看到除了子索引为 0x00 的子 object 外，只有一个子索引为 0x01 的子 object，因此 0x00 的子 object 的值应设置为 1。

EPL_OBD_SUBINDEX_RAM_USERDEF(0x6000, 0x01, kEplObdTypUInt8, kEplObdAccVPR, tEplObdUnsigned8, Sendb1, 0x0)是第二个子 object，其中 0x6000 是索引; 0x01 是子索引; kEplObdTypUInt8 是对象类型，无符号 8 bits；kEplObdAccVPR 是访问类型，kEplObdAccVPR 的意思是变量,可映射为 PDO，只读的； tEplObdUnsigned8 是数据类型，无符号 8 bits； Sendb1 是 object 的名字； 0x0 是默认值。

了解了上面的规则，我们就可以定义自己的object ,例如，我们要定义一个索引为0x6300的object， 他有4个子object。

```
EPL_OBD_BEGIN_INDEX_RAM(0x6300, 0x04, NULL)
    EPL_OBD_SUBINDEX_RAM_VAR(0x6300,    0x00,    kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x3)
    EPL_OBD_SUBINDEX_RAM_USERDEF(0x6300,  0x01,  kEplObdTypUInt8, kEplObdAccVPR, tEplObdUnsigned8, input8, 0x0)
    EPL_OBD_SUBINDEX_RAM_USERDEF(0x6300,  0x02,  kEplObdTypUInt16, kEplObdAccVPRW, tEplObdUnsigned16, output16, 0x0)
    EPL_OBD_SUBINDEX_RAM_USERDEF(0x6300,  0x03,  kEplObdTypUInt32, kEplObdAccVPRW, tEplObdUnsigned32, output32, 0x0)
EPL_OBD_END_INDEX(0x6300)
```

EPL_OBD_BEGIN_INDEX_RAM(0x6300, 0x04, NULL)表示该 object 的索引为 0x6300，一共有 4 个子 object。

EPL_OBD_SUBINDEX_RAM_VAR(0x6300, 0x00, kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x3) 子索引为0x00的子object，默认值为3，表示下面有3个有效的子object。

EPL_OBD_SUBINDEX_RAM_USERDEF(0x6300, 0x01, kEplObdTypUInt8, kEplObdAccVPR, tEplObdUnsigned8, input8, 0x0) 子索引为0x01的子object，数据类型为无符号8 bits，访问类型为只读的，且可映射为PDO。

EPL_OBD_SUBINDEX_RAM_USERDEF(0x6300, 0x02, kEplObdTypUInt16, kEplObdAccVPRW, tEplObdUnsigned16, output16, 0x0) 子索引为0x02的子object，数据类型为无符号16 bits，访问类型（kEplObdAccVPRW）为可读且可写的，且可映射为PDO。

EPL_OBD_SUBINDEX_RAM_USERDEF(0x6300, 0x03, kEplObdTypUInt32, kEplObdAccVPRW, tEplObdUnsigned32, output32, 0x0) 子索引为0x03的子object，数据类型为无符号32 bits，访问类型为可读可写的，且可映射为PDO。

最后，需要注意，objdict.h 中定义的对象字典，被分为三部分：

0x1000 – 0x1FFF : 为 POWERLINK 通信行规区，这些 object 用来保存 POWERLINK 通信所需的参数，如循环周期（0x1006）等。这部分参数被包含在 objdict.h 中的宏定义 EPL_OBD_BEGIN_PART_GENERIC () 和 EPL_OBD_END_PART ()之间。为了保证 POWERLINK 通信，这部分 object 不能随意改动，而且也无需改动。

0x2000 – 0x5FFF :为设备制造商行规，这些 object 用来用来保存设备商自己定义的设备参数。这部分参数被包含在 objdict.h 中的宏定义 EPL_OBD_BEGIN_PART_MANUFACTURER () 和 EPL_OBD_END_PART ()之间。这些 object 需要设备制造商自行定义。

0x6000 – 0x9FFF :为设备行规，这些 object 用来用来保存某一类设备所需要的参数。例如定义一个伺服驱动器，DSP402 中对一个伺服驱动器应该具有哪些参数，这些参数对应的索引和子索引都做了规定。如控制字为 0x6040,状态字为 0x6041。这部分参数被包含在 objdict.h 中的宏定义 EPL_OBD_BEGIN_PART_DEVICE () 和 EPL_OBD_END_PART ()之间。这些 object 需要设备制造商自行定义或者根据 CANopen 的行规添加。

因此，在开发基于 POWERLINK 的产品过程中，当需要添加自定义的 object 时，根据索引的值，将其添加到相应的部分，而且在 objdict.h 文件中的 object， 他们的 index 是按照增序排列的。例如，要添加 index 为 0x5000 的 object，这个 object 需要添加在 0x6000 – 0x9FFF 区间，而且该 object 之前不能有索引比 0x5000 大的 object，否则编译会出错。

5.3 XDD 文件

什么是 XDD 文件？它有什么用处？

简单讲，XDD 文件就是用来描述对象字典的电子说明文档，是 XML DEVICE DESCRIPTION 的简写。设备生产商在自己的设备中实现了对象字典，该对象字典存储在设备里，因此设备提供商需要向设备使用者提供一个说明文档，让使用者知道该设备有哪些参数，以及这些参数的属性。XDD 文件的内容要与对象字典的内容一一对应，即在对象字典中实现了哪些参数，那么在 XDD 文件中就应该有这些参数的描述。

XDD 文件的格式如下图 5-2 所示：

```
<Object index="2201" name="Virtual Outputs 32 Bit" objectType="8">
  <SubObject subIndex="00" name="Number of elements" objectType="7" dataType="0005" accessType="ro" defaultValue="8" PDOmapping="no" />
  <SubObject subIndex="01" name="DWORD 1" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="02" name="DWORD 2" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="03" name="DWORD 3" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="04" name="DWORD 4" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="05" name="DWORD 5" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="06" name="DWORD 6" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="07" name="DWORD 7" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
  <SubObject subIndex="08" name="DWORD 8" objectType="7" dataType="0007" accessType="rw" PDOmapping="RPDO" />
</Object>

<Object index="6000" name="Digital Input 8 Bit" objectType="8">
  <SubObject subIndex="00" name="Number of elements" objectType="7" dataType="0005" accessType="ro" defaultValue="1" PDOmapping="no" />
  <SubObject subIndex="01" name="Byte 1" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" />
</Object>
<Object index="6200" name="Digital Ouput 8 Bit" objectType="8">
  <SubObject subIndex="00" name="Number of elements" objectType="7" dataType="0005" accessType="ro" defaultValue="1" PDOmapping="no" />
  <SubObject subIndex="01" name="Byte 1" objectType="7" dataType="0005" accessType="rw" defaultValue="0x0" PDOmapping="RPDO" />
</Object>
```

图 5-2 XDD 文件中的 object

上面描述了 3 个 object，索引为 0x2201 的 object，有 9 个子索引，每个子索引的属性包括名称，对象类型，数据类型，可访问类型，以及是否可以映射为 PDO。

XDD 文件的作用有两个，第一可以让设备的使用者了解设备有哪些参数，以及这些参数的属性；第二个作用也是主要作用，是给网络配置工具使用，以组建和配置网络。

XDD 文件可以是对象字典（OD）的子集，也就是说，假如在对象字典（OD）中实现了某个 object， 但是在 XDD 文件中却没有这个 object 的描述。这造成的结果是，设备里实际存在某个参数，但是使用者却不知道到该参数的任何信息，包括索引，子索引，数据类型，用途等，从而使用者也就无法使用该参数。 设备制造商，往往为了保护的利益，会可以隐藏一些参数信息。例如，设备制造商给某个客户定制化的产品，该客户不希望设备制造商向其他客户开放这些功能。制造商就可以使用此种方法。

对象字典（OD）不可以是 XDD 文件的子集。如果对象字典（OD） 是 XDD 文件的子集，也就是某个参数在对象字典（OD）中没有定义，在但是却在 XDD 文件中有这个参数的描述信息。当客户试图去访问该参数时，就会返回一些错误，因为设备中根本就没有该参数，这会让使用者苦恼。

XDD 文件是设备提供商提供的，设备提供商在完成产品开发后，根据设备的对象字典

（OD）来编写 XDD 文件。 设备提供商在为客户提供设备时，同时要提供 XDD 文件。XDD 文件与对象字典 OD 的关系如图 5-3：

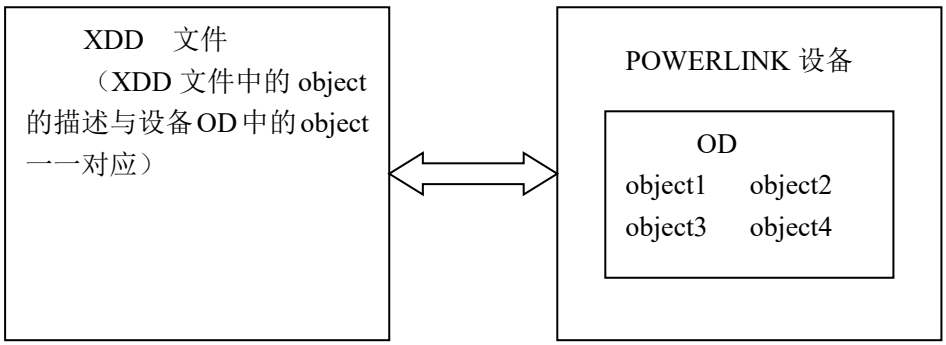


图 5-3 XDD 文件与 OD 的对应关系

对象字典 OD 存在于 POWERLINK 设备中，是设备软件的一部分，XDD 文件是一个 XML 的文件，该文件描述了 POWERLINK 设备中的 object，但该文件不保存在设备中。

下面是一个 XDD 文件的详细说明(可以从 http://www.ethernet-powerlink.org 网站下载 XDD 文件的模板，以及对 XDD 文件格式的详细说明)。

一个 XDD 文件主要有两部分组成：设备描述（Device Profile）和网络通信描述（Communication network profile)。

5.3.1 设备描述 — Device Profile

设备描述包含如下信息：

Device identity： 是描述设备的本身的一些信息。

vendorName：设备生产厂家的名字或者品牌。

vendorID：设备生产厂家的 ID 号。

vendorText：生产厂家的文字描述，例如公司介绍，地址电话等。

deviceFamily：设备类别，标示该设备的种类，例如压力传感器等

productFamily：产品系列，设备制造商自定义的产品系列

productName,：产品名称

productID：产品的 ID

productText：产品的描述

orderNumber：订货号

version：版本描述，包括：软件版本，硬件版本，固件版本

```
<DeviceIdentity>
```

```
            <vendorName>vendor_name</vendorName>
            <!-- todo: replace 12345678 (8 digit hex) with your CANopen vendor ID
in  -->
            <!-- this venor ID is administrated by the CiA (CAN in Automation) -->
            <vendorID>0x12345678</vendorID>
            <!-- todo: Supply additional text about your company/organization -->
            <!-- If unused the element <vendorText> may be deleted -->
            <vendorText>
              <label lang="en">Experiencing problems - contact our support :
++1(..</label>
            </vendorText>
            <!-- todo: Supply information about the family your device belongs to
-->
            <!-- If unused the element <deviceFamily> may be deleted -->
            <deviceFamily>
              <label lang="en">Modular I/O system</label>
            </deviceFamily>
            <!-- todo: replace "MyName" with unique device name as used for file
name -->
            <productName>MyName</productName>
            <!-- todo: replace 1234 with a unique number which identifies the device
type -->
            <!-- If unused the element <productID> may be deleted -->
            <productID>1234</productID>
            <!-- todo: Supply additional text describing your device -->
            <!-- If unused the element <productText> may be deleted -->
            <productText>
              <label lang="en">POWERLINK Dummy I/O device</label>
            </productText>
            <!-- todo: replace "order_text" with device sesignation or article
number -->
            <!-- If unused the element <orderNumber> may be deleted -->
            <orderNumber>order_text</orderNumber>
            <version versionType="HW">1</version>
```

```
        <version versionType="FW">1</version>
        <version versionType="SW">1</version>
    </DeviceIdentity>
```

5.3.2 网络通信描述（Communication network profile）

网络通信描述包含如下信息：Application layers，Transport layers，Network management。

Application layers：

应用层的描述，主要包括如下信息：identity，DataTypeList，ObjectList.

identity：标识信息。这里的标示信息可以是 DeviceIdentity 的子集。也可以没有标示信息。

DataTypeList：数据类型列表。用一个十六进制的数来表示一个数据类型，这样对于配置工具来说， 很容易识别某个 object 的数据类型。下面是 DataTypeList 的例子。

```
        <DataTypeList>
            <defType dataType="0001"><Boolean/></defType>
            <defType dataType="0002"><Integer8/></defType>
            <defType dataType="0003"><Integer16/></defType>
            <defType dataType="0004"><Integer32/></defType>
            <defType dataType="0005"><Unsigned8/></defType>
            <defType dataType="0006"><Unsigned16/></defType>
            <defType dataType="0007"><Unsigned32/></defType>
            <defType dataType="0008"><Real32/></defType>
            <defType dataType="0009"><Visible_String/></defType>
            <defType dataType="0010"><Integer24/></defType>
            <defType dataType="0011"><Real64/></defType>
            <defType dataType="0012"><Integer40/></defType>
            <defType dataType="0013"><Integer48/></defType>
            <defType dataType="0014"><Integer56/></defType>
            <defType dataType="0015"><Integer64/></defType>
            <defType dataType="000A"><Octet_String/></defType>
            <defType dataType="000B"><Unicode_String/></defType>
            <defType dataType="000C"><Time_of_Day/></defType>
            <defType dataType="000D"><Time_Diff/></defType>
```

```
        <defType dataType="000F"><Domain/></defType>
        <defType dataType="0016"><Unsigned24/></defType>
        <defType dataType="0018"><Unsigned40/></defType>
        <defType dataType="0019"><Unsigned48/></defType>
        <defType dataType="001A"><Unsigned56/></defType>
        <defType dataType="001B"><Unsigned64/></defType>
        <defType dataType="0401"><MAC_ADDRESS/></defType>
        <defType dataType="0402"><IP_ADDRESS/></defType>
        <defType dataType="0403"><NETTIME/></defType>
      </DataTypeList>
```

ObjectList：主要包含包含了对 object 和 sub object 的描述。这部分内容要和对象字典（OD）相对应，这里是对设备的对象字典（OD）中的 object 的描述。

举例说明：

下面是一个 object 在 OD 中的定义，以及在 XDD 文件中对该 object 的描述：

在 OD 中的定义：

```
// output variables of master
        EPL_OBD_BEGIN_INDEX_RAM(0x2000, 0x05, NULL)
            EPL_OBD_SUBINDEX_RAM_VAR(0x2000,    0x00,    kEplObdTypUInt8,
kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x4)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x2000,    0x01,    kEplObdTypUInt8,
kEplObdAccVPR, tEplObdUnsigned8, Sendb1, 0x0)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x2000,    0x02,    kEplObdTypUInt8,
kEplObdAccVPR, tEplObdUnsigned8, Sendb2, 0x0)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x2000,    0x03,    kEplObdTypUInt8,
kEplObdAccVPR, tEplObdUnsigned8, Sendb3, 0x0)
            EPL_OBD_SUBINDEX_RAM_USERDEF(0x2000,    0x04,    kEplObdTypUInt8,
kEplObdAccVPR, tEplObdUnsigned8, Sendb4, 0x0)
        EPL_OBD_END_INDEX(0x2000)
```

详细解释：

EPL_OBD_BEGIN_INDEX_RAM(0x2000, 0x05, NULL)：索引为0x2000的object 一共有0x05个sub object。

EPL_OBD_SUBINDEX_RAM_VAR(0x2000, 0x00, kEplObdTypUInt8, kEplObdAccConst, tEplObdUnsigned8, number_of_entries, 0x4):索引为0x2000,子索引为0x00的object表示一共有多少个有效的sub object,这里的值为0x4,表示一共有4个有效的sub object。接下来分别是子索引为0x01，0x02，0x03，0x04的sub object。详细解析如下object。

EPL_OBD_SUBINDEX_RAM_USERDEF(0x2000, 0x01, kEplObdTypUInt8, kEplObdAccVPR, tEplObdUnsigned8, Sendb1, 0x0)

0x2000：object的索引值，16bit无符号整数

0x01 ：object的子索引值，8 bit无符号整数

kEplObdTypUInt8：对象的数据类型（kEplObdTypUInt8表示无符号8bit整数）

kEplObdAccVPR：访问类型

tEplObdUnsigned8：对象数据类型的c语言定义

Sendb1：SDO 的回调函数，当该object被SDO操作时，会执行该回调函数，如果该值北设置为NULL,表示不执行回调函数

0x0 ：该object的默认值

该 object 在 XDD 文件中的描述：

```
<Object index="2000" name="Digital Input 8 Bit" objectType="8">
<SubObject subIndex="00" name="Number of elements" objectType="7" dataType="0005" accessType="ro" defaultValue="4" PDOmapping="no" />
<SubObject subIndex="01" name="Byte 1" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" />
<SubObject subIndex="02" name="Byte 2" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" />
<SubObject subIndex="03" name="Byte 3" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" />
<SubObject subIndex="04" name="Byte 4" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" />
</Object>
```

解释：

```
<Object index="2000" name="Digital Input 8 Bit" objectType="8">
    ......
</Object>
```

这是一个 object 的框架，中间的“……”代表该 object 的 子 object 的描述。每个字段的描述如下：

index="2000"：该object的索引为0x2000（注意这里的数值为16进制）

name="Digital Input 8 Bit"：该object的名字为“Digital Input 8 Bit”,该属性为字符串。

objectType="8"：表示对象类型为数组。该字段的取值为"7"，表示对象为变量；该字段的取值为"8"，表示对象为数组ARRAY；该字段的取值为"9"，表示对象为数组RECORD。

<SubObject subIndex="01" name="Byte 1" objectType="7" dataType="0005" accessType="ro" PDOmapping="TPDO" />

这是索引为0x2000的object下的一个子object定义：

subIndex="01"：该子object的子索引为（注意这里的数值为16进制）

name="Byte 1"：该子object的名字为 “Byte 1”，该属性为字符串

objectType="7"：表示对象类型为变量。该字段的取值为"7"，表示对象为变量；该字段的取值为"8"，表示对象为数组ARRAY；该字段的取值为"9"，表示对象为数组RECORD。

dataType="0005"：数据类型值为"0005"，表示是无符号8bit；用4位16进制数表示对象的数据类型。详细信息如下：

<DataTypeList>

<defType dataType="0001"><Boolean/></defType>

<defType dataType="0002"><Integer8/></defType>

<defType dataType="0003"><Integer16/></defType>

<defType dataType="0004"><Integer32/></defType>

<defType dataType="0005"><Unsigned8/></defType>

<defType dataType="0006"><Unsigned16/></defType>

<defType dataType="0007"><Unsigned32/></defType>

<defType dataType="0008"><Real32/></defType>

<defType dataType="0009"><Visible_String/></defType>

<defType dataType="0010"><Integer24/></defType>

<defType dataType="0011"><Real64/></defType>

<defType dataType="0012"><Integer40/></defType>

<defType dataType="0013"><Integer48/></defType>

<defType dataType="0014"><Integer56/></defType>

<defType dataType="0015"><Integer64/></defType>

<defType dataType="000A"><Octet_String/></defType>

```
<defType dataType="000B"><Unicode_String/></defType>
<defType dataType="000C"><Time_of_Day/></defType>
<defType dataType="000D"><Time_Diff/></defType>
<defType dataType="000F"><Domain/></defType>
<defType dataType="0016"><Unsigned24/></defType>
<defType dataType="0018"><Unsigned40/></defType>
<defType dataType="0019"><Unsigned48/></defType>
<defType dataType="001A"><Unsigned56/></defType>
<defType dataType="001B"><Unsigned64/></defType>
<defType dataType="0401"><MAC_ADDRESS/></defType>
<defType dataType="0402"><IP_ADDRESS/></defType>
<defType dataType="0403"><NETTIME/></defType>
</DataTypeList>
```

accessType="ro"：object的访问类型，可取的值为:

" const " - 只读;该值为常量，数据值不能被改变

" ro " - 只读

" wo " - 只写

" rw "- 可读写

PDOmapping="TPDO"：表示可映射为发送PDO。该属性可设置的值如下：

"no" - 不能映射到PDO

"default" - 根据默认值映射

"TPDO" - 只能被映射到 TPDO

"RPDO" - 只能被映射到 RPDO

"optional" - 可选的，既可以被映射为TPDO，也可以映射RPDO

设备供应商需要根据自己设备的对象字典，手动编辑 XDD 文件。可以使用任何一个可以编辑 XML 文件的工具，打开 XDD 文件进行添加、删除、修改等操作。

从 POWERLINK 的官方网站上 http://www.ethernet-powerlink.org 可以下载到 XDD 文件的模板和格式（Powerlink_Main.xsd）说明。Powerlink_Main.xsd 描述了 POWERLINK 的 XDD 文件的结构和语法。可以使用该文件来验证 XDD 文件，以检查该 XDD 文件是否符合要求。检查的内容包括文档中出现的元素、文档中出现的属性、子元素、子元素的数量、子元素的顺序、元素是否为空、元素和属性的数据类型、元素或属性的默认和固定值。XML Schema 本身是一个 XML 文档，它符合 XML 语法结构。可以用通用的 XML 解析器解析它。

5.4 异步通信模型-SDO

SDO 是 Service Data Object 的缩写。SDO 和 PDO 一样，是通信数据对象，也就是在网络上收发的数据。SDO 和 PDO 的区别在于：PDO ： 用来周期性地传输过程数据，用于同步通信; 而 SDO 传输的是异步数据，即非周期的，偶尔传输的数据，它用来传输网络命令，配置网络参数，以及偶尔对其它节点的 object 的访问等。

对于 SDO 应用最多的有两个操作：

5.4.1 SDO 读

访问另一个节点的 object 时，该节点可以向另一个节点发送 SDO 读命令，来访问另一个节点的 object。被访问的节点，收到了 SDO 读命令以后，会把相应的数据发送到网络上，传给需要的节点。

```
tEplKernel PUBLIC EplApiReadObject(
            tEplSdoComConHdl* pSdoComConHdl_p,
            unsigned int      uiNodeId_p,
            unsigned int      uiIndex_p,
            unsigned int      uiSubindex_p,
            void*             pDstData_le_p,
            unsigned int*     puiSize_p,
            tEplSdoType       SdoType_p,
            void*             pUserArg_p)
```

函数描述: 通过SDO 的方式读指定节点上的指定的object。如果是远程节点（非本节点），该函数会产生一次SDO的传输，发送一个SDO 的读命令给远程节点，当远程节点返回数据时,POWERLINK协议栈会发送一个消息kEplApiEventSdo,同时执行AppCbEvent回调函数。

参数描述:

pSdoComConHdl_p = INOUT: SDO 连接的句柄，是一个指针。如果读本地的OD时，该指针可以为空。否则该指针指向一个unsigned int 的变量。

uiNodeId_p = IN: node ID (0 ：表示本地)，要访问的节点的node Id。

uiIndex_p = IN: 要访问的object的索引值

uiSubindex_p = IN: 要访问的object的子索引值

pDstData_le_p = OUT: 数据缓冲区，读到的数据被存放到该指针指向的数据区中(以小端排列)。

puiSize_p = INOUT: 要读的object的大小，以字节为单位。这个参数是指针，指向一个变量。

SdoType_p = IN: SDO传输的类型。

pUserArg_p = IN: 用户定义指针，该指针会被传递到事件回调函数。

例程如下：

```
unsigned int SdoComConHdl;
EplRet = EplApiReadObject(&SdoComConHdl, m_uiNodeId, 0x1006, 0x00, &dw_le_CycleLen_g, 4, kEplSdoTypeAsnd, NULL);
```

5.4.2 SDO 写

```
tEplKernel PUBLIC EplApiWriteObject(
    tEplSdoComConHdl* pSdoComConHdl_p,
    unsigned int      uiNodeId_p,
    unsigned int      uiIndex_p,
    unsigned int      uiSubindex_p,
    void*             pDstData_le_p,
    unsigned int*     puiSize_p,
    tEplSdoType       SdoType_p,
    void*             pUserArg_p)
```

参数描述:

SdoComConHdl_p = INOUT: SDO 连接的句柄，是一个指针。如果读本地的OD时，该指针可以为空。否则该指针指向一个unsigned int 的变量。

uiNodeId_p = IN: node ID (0 ：表示本地)，要访问的节点的node Id。

uiIndex_p = IN: 要访问的object的索引值

uiSubindex_p = IN: 要访问的object的子索引值

pDstData_le_p = OUT: 数据缓冲区，读到的数据被存放到该指针指向的数据区中(以小端排列)。

puiSize_p = INOUT: 要访问的object的大小，以字节为单位。这个参数是指针，指向一个变量。

SdoType_p = IN: SDO传输的类型。

pUserArg_p = IN: 用户定义的参数指针，该指针会被传递到事件回调函数。

EplRet = EplApiWriteObject(&SdoComConHdl , m_uiNodeId, 0x1006, 0x00, &dw_le_CycleLen_g, 4, kEplSdoTypeAsnd, NULL);

SDO 读写操作完成后，协议栈会触发事件回调函数（AppCbEvent（））, 触发事件回调函数的事件为 kEplApiEventSdo （该事件的定义在 Epl.h 中）。SDO 的读写的参数信息和结果保存在 pEventArg_p->m_Sdo 中。pEventArg_p->m_Sdo 的数据结构如下所示：

```
typedef struct
{
    tEplSdoComConHdl    m_SdoComConHdl;
    tEplSdoComConState  m_SdoComConState;
    DWORD               m_dwAbortCode;
    tEplSdoAccessType   m_SdoAccessType;
    unsigned int        m_uiNodeId;         // NodeId of the target
    unsigned int        m_uiTargetIndex;    // index which was accessed
    unsigned int        m_uiTargetSubIndex; // subindex which was accessed
    unsigned int        m_uiTransferredByte; // number of bytes transferred
    void*                 m_pUserArg;           // user definable argument
pointer
```

```
} tEplSdoComFinished;
```

下面是一个处理该事件的例子:

1. 首先调用 EplApiWriteObject 或 EplApiReadObject 函数

```
EplRet = EplApiWriteObject(&SdoComConHdl, m_uiNodeId, 0x1006, 0x00,
&dw_le_CycleLen_g, 4, kEplSdoTypeAsnd, NULL);
```

2. 检查事件回调函数AppCbEvent（）中是否有对“kEplApiEventSdo”消息的处理。

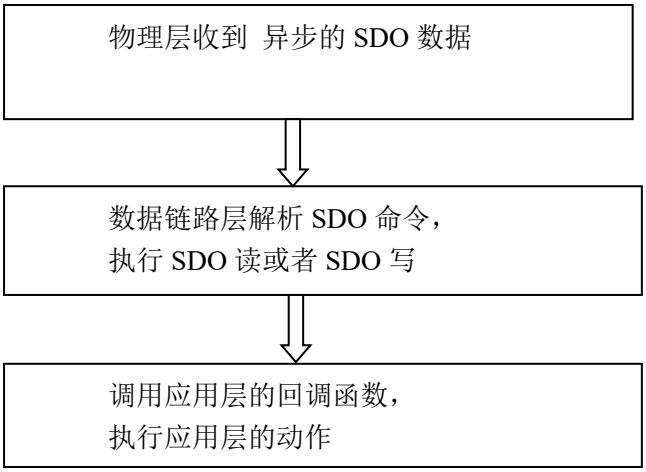
```
EplKernel PUBLIC AppCbEvent(
    tEplApiEventType        EventType_p,    // IN: event type (enum)
    tEplApiEventArg*        pEventArg_p,    // IN: event argument (union)
    void GENERIC*           pUserArg_p)
{
tEplKernel          EplRet = kEplSuccessful;

    UNUSED_PARAMETER(pUserArg_p);

    // check if NMT_GS_OFF is reached
    switch (EventType_p)
    {
        ..................................................
..................................................
  case kEplApiEventSdo:
        {   // SDO transfer finish
      if (pEventArg_p->m_Sdo.m_SdoComConState == kEplSdoComTransferFinished)
            {
                if(pEventArg_p->m_Sdo.m_dwAbortCode == 0)
                 {   // ... successfully
                pEventArg_p->m_Sdo.m_uiNodeId //Sdo命令的node id
```

```
                    pEventArg_p->m_Sdo.m_uiTargetIndex//Sdo命令的索引
                    pEventArg_p->m_Sdo.m_uiTargetSubIndex Sdo命令的子索引
                        }
                }
            }// 对kEplApiEventSdo事件处理完毕
                    ……..
                    ……..
}//事件回调函数AppCbEvent（）处理完毕
```

更详细的说明，请见 openPOWERLINK 源码中 Documentation 文件夹中的文档。

5.4.3 SDO 命令接收方的处理过程

上述为 SDO 命令发送方的处理过程，对于 SDO 命令的接收方，处理的过程如下：



对于使用者，需要关心在哪里设置 SDO 命令的回调函数， 因为当 POWERLINK 的站点收到 SDO 的读命令或者 SDO 的写命令时，是通过回调函数通知应用层的。应用层需要在回调函数里执行一下操作。SDO 的读命令或者写命令是对目标节点中指定索引和子索引的

object 进行操作。因此回调函数应该与某个 object 相关。在对象字典 OD 中，声明某个 object 时，有一个参数用来指定回调函数，将回调函数的名称写在这里，如下所示的索引为 0x1600 的 object，其回调函数的名字为 EplPdouCbObdAccess。这一步操作，完成的是将名称为 EplPdouCbObdAccess 的回调函数与 0x1600 的 object 绑定。但是 EplPdouCbObdAccess 这个函数完成什么功能，还没有定义。

```
EPL_OBD_BEGIN_INDEX_RAM(0x1600, 0x02, EplPdouCbObdAccess)
        EPL_OBD_SUBINDEX_RAM_VAR(0x1600,  0x00,  kEplObdTypUInt8,
kEplObdAccRW, tEplObdUnsigned8, NumberOfEntries, 0x01)
        EPL_OBD_SUBINDEX_RAM_VAR(0x1600, 0x01, kEplObdTypUInt64,
    EPL_OBD_END_INDEX(0x1600)
```

对函数 EplPdouCbObdAccess 的声明和定义，在本示例中在源文件 EplPdou.c 和 EplPdou.h 中。用户自定义的回调函数，使用者需要根据自己的实际需要，在相应的文件中声明，并编写该函数的功能。

```
tEplKernel PUBLIC EplPdouCbObdAccess(tEplObdCbParam MEM* pParam_p)
{
tEplKernel          Ret = kEplSuccessful;
unsigned int        uiIndexType;
BYTE                bMappObjectCount;
tEplObdAccess       AccessType;
unsigned int        uiCurPdoSize;
    pParam_p->m_dwAbortCode = 0;
    if (pParam_p->m_ObdEvent != kEplObdEvPreWrite)
    {   // read accesses, post write events etc. are OK
```

```
            goto Exit;
        }
    Exit:
        return Ret;
    }
```

在 openPOWERLINK 中，已经定义好的用户的异步回调函数为 EplApiCbObdAccess，用户可以直接使用，使用时将该函数在 OD 中与相应的 object 绑定，并在 EplApiCbObdAccess 函数中增加相应的处理。回调函数的参数如下：

```
typedef struct
{
    tEplObdEvent    m_ObdEvent; //对该函数的解释见下文
    unsigned int    m_uiIndex; // 收到 SDO 命令的索引
    unsigned int    m_uiSubIndex; // 收到 SDO 命令的子索引
    void *          m_pArg;
    DWORD           m_dwAbortCode; // 收到 SDO 命令的错误码

} tEplObdCbParam;
```

下面是对回调函数中参数 m_ObdEvent 的 解释：

```
typedef enum
{
    kEplObdEvCheckExist         = 0x06,     // 检查对象是否存在
    kEplObdEvPreRead            = 0x00,     // 在读一个 object 在 OD 中的数据之前触发的
```

事件。

kEplObdEvPostRead = 0x01, //在读一个object在OD中的数据之后触发的事件。

kEplObdEvWrStringDomain = 0x07, // 改变字符串或者域数据指针或大小时触发的事件。

kEplObdEvInitWrite = 0x04, // 检查要写入的对象的大小。

kEplObdEvPreWrite = 0x02, //在写一个object在OD中的数据之前触发的事件。

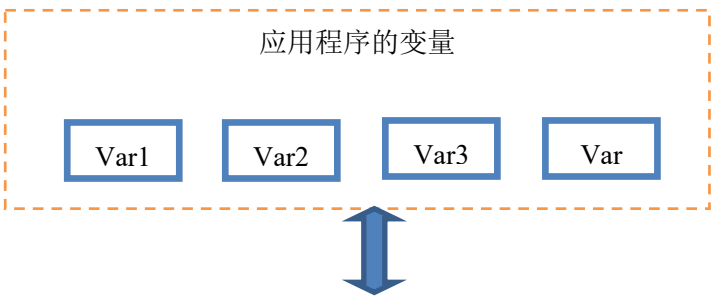
kEplObdEvPostWrite = 0x03, // 在写一个object在OD中的数据之后触发的事件。

// kEplObdEvAbortSdo = 0x05 // 在传输中止之后触发的事件

} tEplObdEvent;

5.5 同步通信模型-PDO

同步通信即周期性的实时通信，用来传输过程数据对象（PDO），传输的模型如下图 5-4 所示。首先 POWERLINK 站点从物理层接收到数据帧， 数据帧中包含了过程数据对象（PDO）。POWERLINK 协议栈根据接收 PDO 的映射关系来解析数据帧，将接解析的数据存放到 OD 中相应的 object 指向的应用程序的变量中 。对于发送过程， 是本过程的逆过程，读者试着自己理解。过程数据对象（PDO）D 的传输过程如图 5-4 所示。



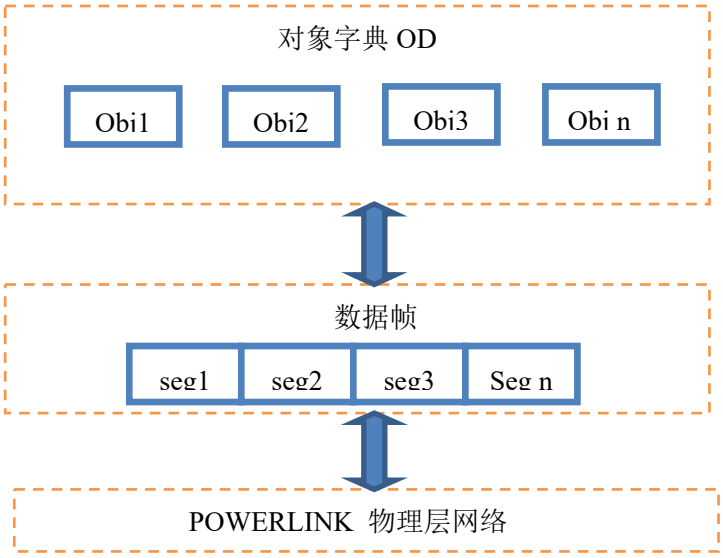


图 5-4 PDO 数据的处理过程

在 CANopen 的协议中，每个节点都有两种参数：通信参数和映射参数。

通信参数：决定该节点需要接收来自哪个节点的数据，或者将数据发送给哪个节点。对于从节点，由于发送数据帧是广播的，因此不需要设置该参数；对于主站的发送数据帧 Preq，需要设置该参数，来标示该数据帧是发送给哪个从节点的。

映射参数：决定该节点如何组成要发送的数据包或者如何解析收到的数据包。也就是确定对象字典中的对象（Object）与数据包中数据段的对应关系。

每个 POWERLINK 节点接收和发送数据时，都有一组通信参数和映射参数来描述。

对于接收：

0x1400-0x14ff：为通信参数

0x1600-0x16ff：为映射参数

通信参数和映射参数成对出现，一一对应。0x1400 与 0x1600 为一对；0x1401 与 0x1601 为一对；0x14xx 与 0x16xx 为一对。

对于发送：

0x1800-0x18ff：为通信参数

0x1A00-0x1Aff：为映射参数

通信参数和映射参数成对出现，一一对应。0x1800 与 0x1A00 为一对；0x1801 与 0x1A01 为一对；0x18xx 与 0x1Axx 为一对。

5.5.1 从站发送配置之通信参数配置（0x18XX）

参数 0x18XX 就是描述发送配置的通信参数，XX 的取值范围为 0x00 至 0xFF。

该参数描述此节点需要把自己的数据帧发送给网络中的哪个节点（根据节点号区分）。从我们前面讲的 POWERLINK 基本原理可知，每个从站在上报自己数据的时候，都是以广播的形式发送，也就是说从站发送自己数据的时候，发送的目标地址是 255。因此所有从节点的发送数据的通信参数都是 255。

一个 POWERLINK 数据帧最大为 1500 个字节，节点在发送数据时，可以把很多参数打包成一个大的数据帧。所以在 POWERLINK 网络中，一个从节点每个循环周期只需要发送一个数据帧就够了。因此对于从节点，我们只需要实现并使用 0x1800 就够了。

5.5.2 从站发送配置之映射参数配置（0x1A00）

由于通信参数和映射参数成对出现，一一对应。0x1800 与 0x1A00 为一对；因此对于从节点，当你使用 0x1800 作为通信参数时，那么你就必须使用 0x1A00 作为映射参数。映射参数解决的是对象字典中的对象 Object（即参数）与数据帧中数据段的对应关系。对于发送来说，也就是如何将要发送的 Object 组成一个数据帧。

如图 5-5，描述了一个数据帧，和一个对象字典中需要发送的对象。现在需要将这些要发送的参数，组成一个数据帧。也就是把每一个要发送的 Object 与某一个数据段建立映射关系,把 Object 的值放到数据帧中对应的字段。数据帧中的字段与 OD 中的 object 的对应关系如图 5-5 所示。

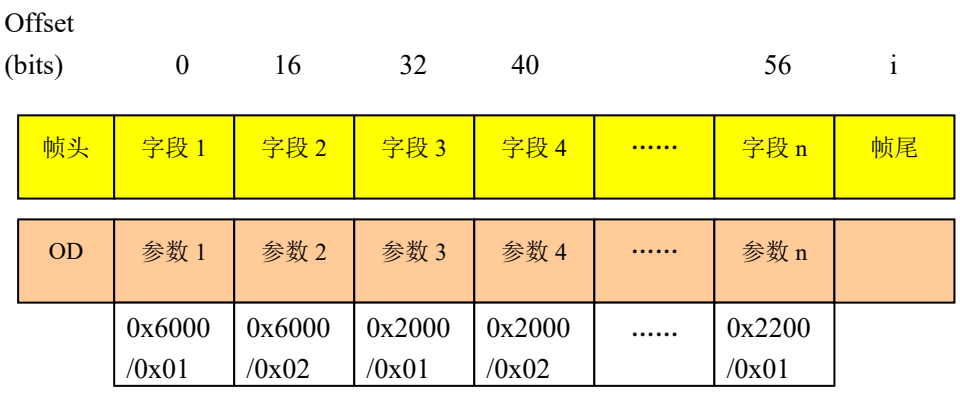


图 5-5 数据帧与 OD

假如我们想建立如图 5-6 所示的映射关系：

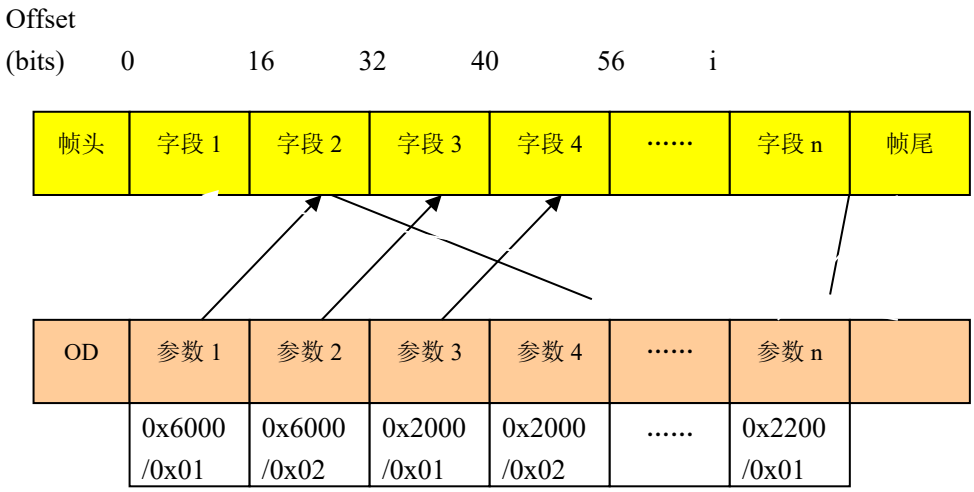


图 5-6 数据帧中的字段与 OD 中的 object 的对应关系

即要发送的数据在数据帧中的位置：

对象 0x2000/02 放在数据帧偏移量为 0 的地方，长度为 16bits；

对象 0x6000/01 放在数据帧偏移量为 16 的地方，长度为 16bits；

对象 0x6000/02 放在数据帧偏移量为 32 的地方，长度为 8bits；

对象 0x2000/01 放在数据帧偏移量为 40 的地方，长度为 16bits。

为此，我们需要将 0x1A00 的值配置如下：

0x1A00/0x01 值： 0x0010000000022000

0x1A00/0x02 值： 0x0010001000016000

0x1A00/0x03 值： 0x0008002000026000

0x1A00/0x04 值： 0x0010002800022000

下面来详细解释一下第一句话的意思，“0x1A00/0x01 值 0x0010000000022000”

0x1A00/0x01 是用来描述第一个 Object 与数据帧中字段的映射关系的参数，把他的值设为 0x0010000000022000，意思是将索引为 0x2000 子索引为 0x02 的对象映射到数据帧偏移量为 0x0000（数据帧开头)的地方，长度为 16bits。

0x 0010 0000 00 02 2000

长度（bits） 偏移量（bits） 保留 object 的子索引 object 的子索引

5.5.3 从站接收配置之通信参数配置（0x14XX）

参数 0x14XX 描述接收配置的通信参数，XX 的取值范围为 0x00 至 0xFF。

该参数描述了此节点需要接收来自哪个节点的数据。从前面讲述的 POWERLINK 基本原理可知，POWERLINK 支持交叉通信，因此每一个节点都可以接收来自另外一个或多个节点的数据。所以一个节点可以有多个接收通道。例如 0x1400 是一个通道，接收来自主节点的数据，那么就把 0x1400/0x01 的值设为 0（默认值设为 0，表示接收来自主站的请求数据）；0x1401 是一个通道，接收来自 3 号节点的数据，那么就把 0x1401/0x01 的值设为 3，这样该节点在同一个循环周期你既接收来自主站的数据，也接收来自 3 号节点的数据。

5.5.4 从站接收配置之映射参数配置（0x1600）

由于通信参数和映射参数成对出现，一一对应。0x1400 与 0x1600 为一对，0x1401 与 0x1601 为一对…；因此对于从节点，当你使用 0x1400 作为通信参数时，那么你就必须使用 0x1600 作为映射参数。映射参数解决的是对象字典中的对象 Object（即参数）与数据帧中数据段的对于关系。对于接收来说，也就是如何将一个数据帧进行解析。因为某个节点发来的数据，往往是多个 Object 打包在一起的，接收方需要知道自己应该接收数据帧的哪一段或者哪几段，数据长度多长。

如图 5-7，描述了一个数据帧，和对象字典中的对象。现在需要将收到的数据帧中的数据段和对象字典中的 Object 建立映射关系，如图 5-7 所示。

Offset (bits)	0	16	32	40		56	i
帧头	字段 1	字段 2	字段 3	字段 4	……	字段 n	帧尾
OD	参数 1	参数 2	参数 3	参数 4	……	参数 n	
	0x6000 /0x01	0x6000 /0x02	0x2000 /0x01	0x2000 /0x02	……	0x2200 /0x01	

图 5-7 数据帧与 OD

假如我们想建立如图 5-8 所示的映射关系：

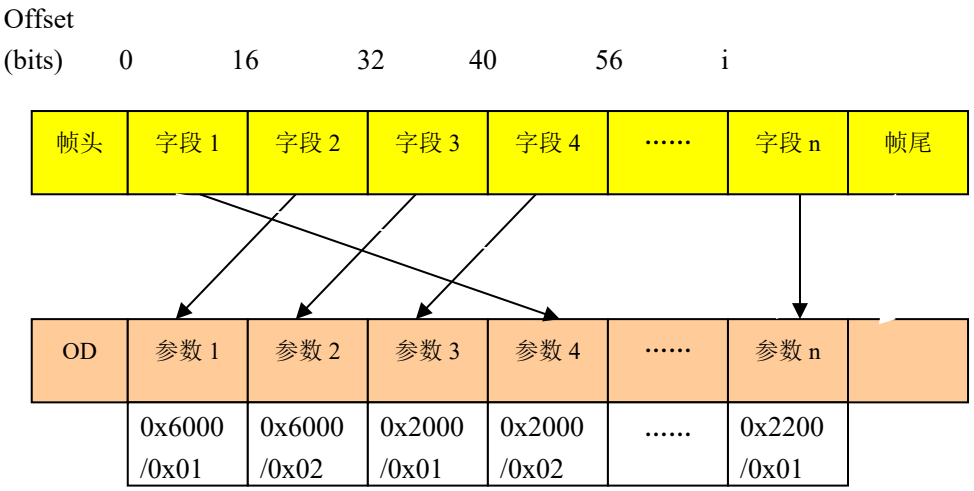


图 5-8 数据帧中的字段与 OD 中的 object 的对应关系

即接收到的数据帧中数据段与 Object 的映射关系：

数据帧偏移量为 0，长度为 16bits，这段数据被截取放到对象 0x2000/02；

数据帧偏移量为 16，长度为 16bits，这段数据被截取放到对象 0x6000/01；

数据帧偏移量为 32，长度为 8bits，这段数据被截取放到对象 0x6000/02；

数据帧偏移量为 40，长度为 16bits，这段数据被截取放到对象 0x2000/01。

为此，我们需要将 0x1600 的值配置如下：

0x1600/0x01 值： 0x0010000000022000

0x1600/0x02 值： 0x0010001000016000

0x1600/0x03 值： 0x0008002000026000

0x1600/0x04 值： 0x0010002800022000

下面来详细解释第二句话的意思，“0x1600/0x02 值 0x0010001000016000”。

0x1600/0x02 是用来描述第二个 Object 与数据帧中字段的映射关系的参数，把他的值设为 0x0010001000016000，意思是数据帧中偏移量为 0x0010（第 16 bit 开始），长度为 0x0010（16bits）的这段数据映射到索引为 0x6000 子索引为 0x01 的对象。

0x	0010	0010	00	01	6000
	长度（bits）	偏移量（bits）	保留	object 的子索引	object 的子索引

5.5.5 主站发送参数的配置过程

主站和从站的区别：每个循环周期，从站只需要发送一个 TPDO 的数据帧。

而主站如果基于请求/应答模式，一个循环周期需要向网络中所有的节点都发送一次请求数据帧 Preq，而且相应的也会收到从站的回复 Pres，一个 Preq 数据帧就是一个 TPDO,而一个 Pres 数据帧，就是一个 RPDO。这也就意味着主站在发送时，需要有多个发送 TPDO 的通道；在接收时，需要有多个接收 RPDO 的通道。举例来说，假如一个系统里，有 1 个主节点和 3 个从节点。此时主站需要 3 个发送通道和 3 个接收通道。

对于主站的发送需要如下配置：

0x1800/0x01 值 1；表示发送数据给 1 号从节点

0x1a00/0x01 值 0x0010000000012000；

0x1a00/0x02 值 0x0010010000022000；

对象 Object 与数据段的映射关系

……..

0x1801/0x01 值 2；表示发送数据给 2 号从节点

0x1a01/0x01 值 0x0010000000016000；

0x1a01/0x02 值 0x0010010000026000；

对象 Object 与数据段的映射关系

……..

0x1803/0x01 值 3；表示发送数据给 3 号从节点

0x1a03/0x01 值 0x0010000000013000；

0x1a03/0x02 值 0x0010010000023000；

对象 Object 与数据段的映射关系

……..

对于接收需要如下配置：

0x1400/0x01 值 1；表示接收来自 1 号从节点的数据

0x1600/0x01 值 0x0010000000012100；

0x1600/0x02 值 0x0010010000022100；

对象 Object 与数据段的映射关系

……..

0x1401/0x01 值 2；表示接收来自 2 号从节点的数据

0x1601/0x01 值 0x0010000000016100；

0x1601/0x02 值 0x0010010000026100；

对象 Object 与数据段的映射关系

……..

0x1403/0x01 值 3；表示接收来自 3 号从节点的数据

0x1603/0x01 值 0x0010000000013100；

0x1603/0x02 值 0x0010010000023100；

对象 Object 与数据段的映射关系

注意：PDO 的配置不支持动态变化， 一旦配置好了网络参数和映射参数，进入到同步通信阶段，网络中的各个节点就按照设定的参数，来组建数据帧以及解析数据帧。

5.6 应用程序接口- API：

应用程序存取通信对象的接口，将要发送的变量赋值给通信对象，要收到的通信对象赋值给相应的应用变量。

在 POWERLINK 的通信过程中，每个节点收发的数据都已存在节点的 object 中，用户的应用程序有两种方法来访问 object 中的数据。

用户可以通过 object 的索引和子索引，检索到该 object，然后向该 object 写入数值，或者从 object 把数值读出来。这个方法效率很低，不适合用于周期性传输的 PDO 数据。

将用户自己定义的变量与本地对象字典中的对象链接起来，这样 POWERLINK 协议栈，会将收到的数据直接存放到用户自己定义的变量；同样将用户自己定义的变量直接放到数据帧中，发送出去。此时，object 将用户定义的变量空间作为自己的数据区。这种方法，使得 POWERLINK 协议栈与用户的应用程序之间的数据交互简单，而且效率很高。

通过调用如下函数将应用程序的变量和object 链接，这里链接的object是被配置用于

同步通信的object。

```
EplRet = EplApiLinkObject(0x6000, &bVarIn1_1, &uiVarEntries, &ObdSize, 0x01);
```

5.7 自定义应用层

无论采用 CANopen 做为应用层，还是自定义应用层，数据链路层都要遵循 POWERLINK 的协议。架构如图 5-9：

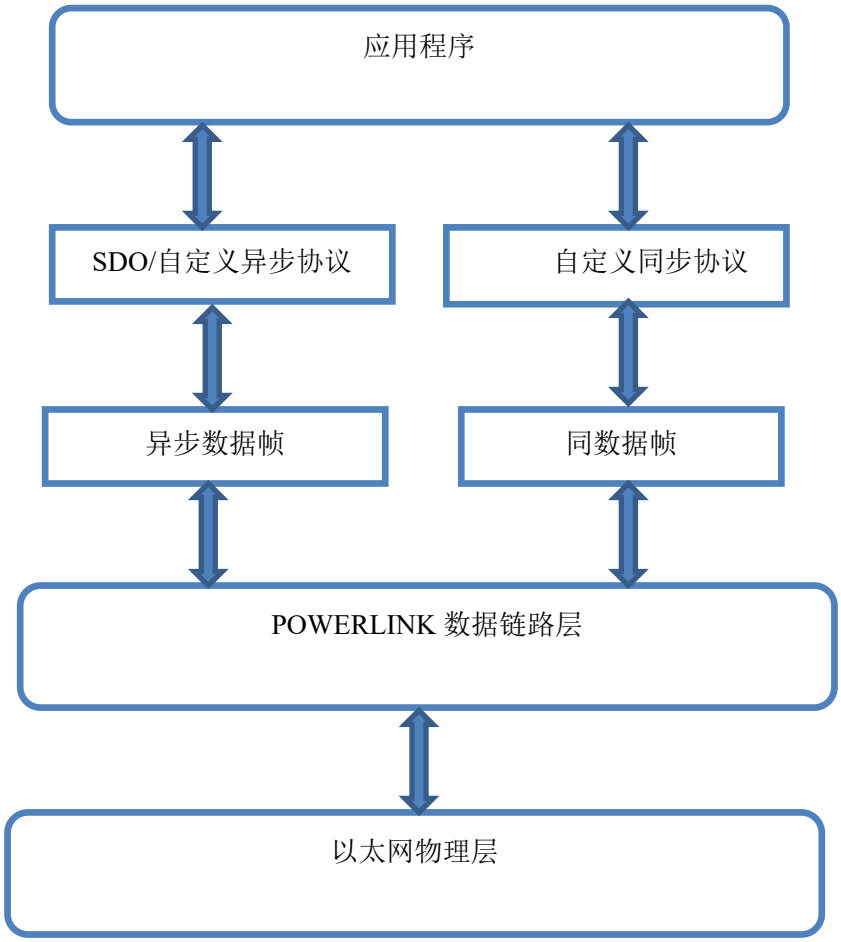


图 5-9 应用层协议模型

5.7.1 对象字典 OD

在自定义的应用层中， 一般不需要对象字典，应用程序通过自定义的异步协议和自定义的同步协议，直接和相应的数据帧交互。

5.7.2 自定义异步协议

用户根据自己的需要处理收到的异步数据帧，组建要发送的异步数据帧。数据帧的格式由用户自己定义，但如果欲与基于 CANopen 的 POWERLINK 设备相连，数据帧的格式需要遵循 CANopen 的 SDO。

5.7.3 自定义同步协议

基于 CANopen 的 PDO，通过网络参数和映射参数，将 OD 中的数据组建数据帧，以及解析数据帧并将数据存放到 OD。 在自定义的同步协议中， 应用程序直接与同步数据帧交换，用户自己决定如何组建数据帧以及如何解析数据帧。

本章介绍了CANopen应用层的原理和机制， 如果用户不想使用CANopen做为应用层，可以定制化自己的应用层。在 openPOWERLINK 上定制自己的应用层软件比较困难，因为要修改的代码较多，涉及的模块也较多。如果在 HDL POWERLINK 上定制化自己的应用层软件，会方便很多， 如何实现， 请参考本书的 HDL POWERLINK 解决方案一章 。基于 HDL POWELRINK 更详细的设计和说明，可以参考 HDL POWERLINK 使用手册。