CANopen协议简介与应用开发

要　点：

- CANopen协议简介；
- CANopen协议具体应用开发。

一　CANopen协议简介

CANopen协议是建立在CAN基础上的通信网络技术，是欧洲CiA组织定义的几种CAN高层协议标准之一，因其开放性和低成本等优势，发布后不久就得到了广大厂商和用户的亲睐。在欧洲，在众多基于CAN的工业系统中，CANopen协议是事实上占据领导地位的标准。目前CANopen协议已经在运动控制、车辆工业、电机驱动、工程机械、船舶海运、楼宇自动化等行业得到广泛的应用。

1、协议特性

◆ 基于多种网络通信模式，适合对时间要求苛刻的通讯场合。

◆ 以对象字典的形式来定义标准设备描述文件，并通过SDO(服务数据对象) 报文存取。

◆ 包含标准的设备监护服务信息（节点监视/心跳）、网络管理信息以及紧急信息对象。

◆ 支持同步、异步传输模式。

◆ 提供预定义连接集，简化系统配置工作。

◆ 在实际应用中，一般以主从式结构模式为主。

1.1 主节点主要的功能

- 控制从节点运行状态；
- 通过SDO对对象字典的参数进行读取和修改；
- 监视从节点的运行状况和是否发生错误(即进行节点保护)；
- 由从节点中获得相应的信息和运行状态；启动和终止SDO通讯；
- 发送SYNC报文；
- 提供分配特定标识符等网络管理的服务。

1.2 从节点功能

- 负责底层的网络通讯和控制任务，每个从节点只完成属于自己范围内的特定任务。
- 进行实时数据的传输，即PDO (过程数据对象)的通讯。
- 响应主节点发送的管理信息帧。
- 对自己负责的底层设备进行数据采集和控制。

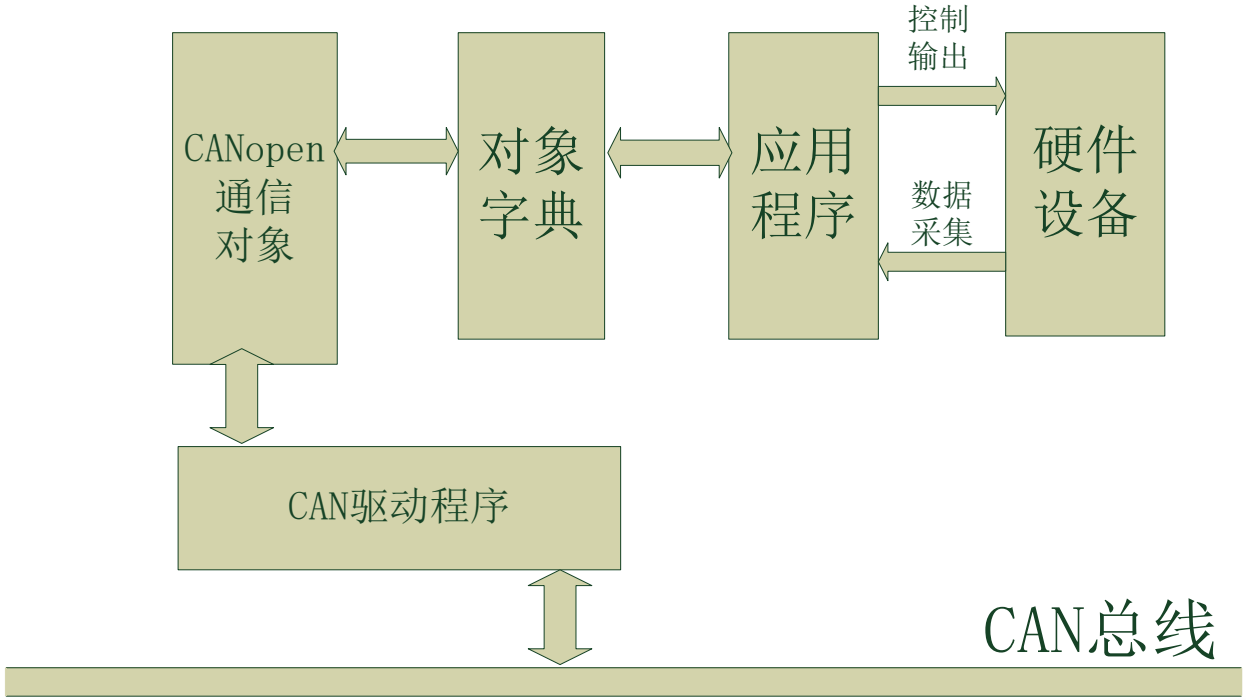
1.3 从节点架构

CAN驱动：CAN控制器初始化，报文收发处理。

通信对象：收发和解析各种报文对象，实现数据交互过程。

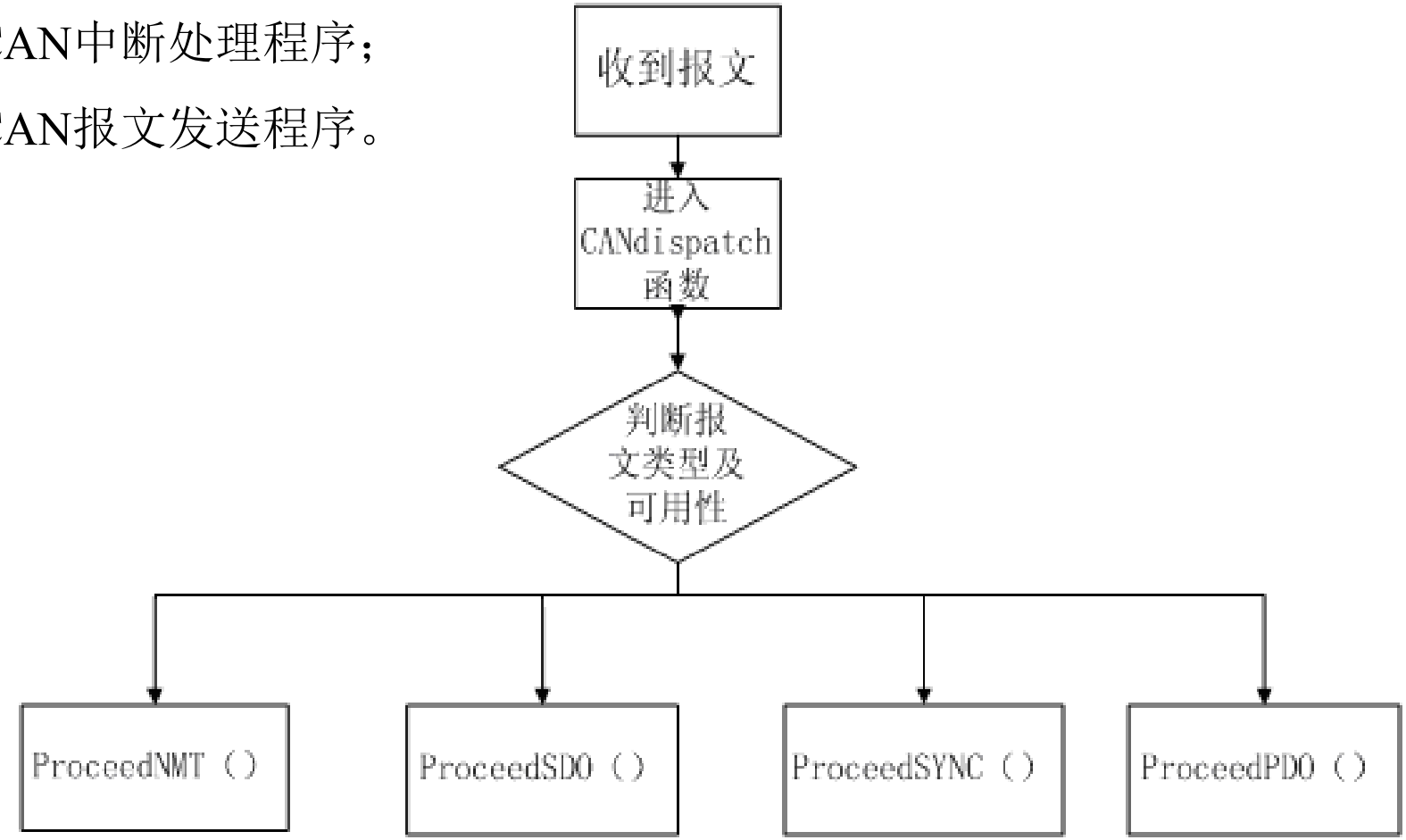
对象字典：存储节点所有信息，向通信对象和应用程序提供接口。

应用程序：实现节点的功能，如控制、数据采集等，用户自定义。



1、 CAN驱动程序

- CAN控制器初始化程序；
- CAN中断处理程序；
- CAN报文发送程序。



2、对象字典

对象字典就是一个有序的对象组，每个对象采用一个16位的索引值来寻址，为了允许访问数据结构中的单个元素，同时也定义了一个8位的索引值，这个索引值通常被称为子索引。

每个CANopen设备都有一个对象字典，对象字典包含了描述这个设备和它的网络行为的所有参数，对象字典通常用电子数据文档（EDS）来记录这些参数。

索引	对象
0000	Not used
0001 - 001F	静态数据类型 （标准数据类型，如 Boolean，Integer 16）
0020 - 003F	复杂数据类型 （预定义由简单类型组合成的结构如 PDOCommPar，SDOParameter）
0040 - 005F	制造商规定的复杂数据类型
0060 - 007F	设备子协议规定的静态数据类型
0080 - 009F	设备子协议规定的复杂数据类型
00A0 - 0FFF	Reserved
1000 - 1FFF	通讯子协议区域 (如设备类型，错误寄存器，支持的 PDO 数量)
2000 - 5FFF	制造商特定子协议区域
6000 - 9FFF	标准的设备子协议区域 (例如“DSP-401 I/O 模块设备子协议”：Read State 8 Input Lines 等)
A000 - FFFF	Reserved

- 通信子协议
  - SYNC报文参数设置（1005H、1006H、1007H）
  - SDO报文参数设置

    客户端SDO（1200H-127FH）

    服务器SDO（1280H-12FFH）
  - 接收PDO报文参数设置

    PDO通信参数（1400H-15FFH）

    PDO映射参数（1600H-17FFH）
  - 发送PDO报文参数设置

    PDO通信参数（1800H-19FFH）

    PDO映射参数（1A00H-1BFFH）
- 设备子协议

  与特定设备相关的参数，用于存储设备的相关参数。（6000H-9FFFH）

- 对象的结构

Index (hex)	Object (Symbolic Name)	Name	Type	Attrib.	M/O

Index： 指出该对象在对象字典中的位置

Object：指明该对象的类型

Name：对象的名称

Type： 指明该对象的数据类型

Attrib： 对象的属性值

M/O： 说明该对象是强制实现的还是可选的

- 从节点对象字典实现

由于从节点需要实现的对象字典项不是很多，也不复杂，所以可以直接采用二维数组的形式来实现。例如：

```
/* index 0x1400 :   Receive PDO 1 Parameter. */
                    UNS8 ObjDict_highestSubIndex_obj1400 = 5; /* number of subindex - 1*/
                    UNS32 ObjDict_obj1400_COB_ID_used_by_PDO = 0x200; /* 512 */
                    UNS8 ObjDict_obj1400_Transmission_Type = 0x1; /* 1 */
                    UNS16 ObjDict_obj1400_Inhibit_Time = 0x0; /* 0 */
                    UNS8 ObjDict_obj1400_Compatibility_Entry = 0x0; /* 0 */
                    UNS16 ObjDict_obj1400_Event_Timer = 0x0;  /* 0 */
                    subindex ObjDict_Index1400[] =
                     {
                       { RO, uint8, sizeof (UNS8), (void*)&ObjDict_highestSubIndex_obj1400 },
                       { RW, uint32, sizeof (UNS32), (void*)&ObjDict_obj1400_COB_ID_used_by_PDO },
                       { RW, uint8, sizeof (UNS8), (void*)&ObjDict_obj1400_Transmission_Type },
                       { RW, uint16, sizeof (UNS16), (void*)&ObjDict_obj1400_Inhibit_Time },
                       { RW, uint8, sizeof (UNS8), (void*)&ObjDict_obj1400_Compatibility_Entry },
                       { RW, uint16, sizeof (UNS16), (void*)&ObjDict_obj1400_Event_Timer }
                     };
```

- 对象字典实现

将所有对象集合起来，得到对象字典；

```
const indextable ObjDict_objdict[] =
{
    {
(subindex*)ObjDict_Index1000,sizeof(ObjDict_Index1000)/sizeof(ObjDic
t_Index1000[0]), 0x1000}, {
(subindex*)ObjDict_Index1001,sizeof(ObjDict_Index1001)/sizeof(ObjDic
t_Index1001[0]), 0x1001}, {
(subindex*)ObjDict_Index1003,sizeof(ObjDict_Index1003)/sizeof(ObjDic
t_Index1003[0]), 0x1003},
    ……
};
```

- 扫描函数

根据对象字典的实现方式，相应的扫描函数实现如下：

```
const indextable * ObjDict_scanIndexOD (UNS16 wIndex, UNS32 * errorCode,
    ODCallback_t **callbacks)
{
            int i;
           *callbacks = NULL;
            switch(wIndex){
            case 0x1000: i = 0;break;
            case 0x1001: i = 1;break;
            case 0x1003: i = 2;*callbacks = ObjDict_Index1003_callbacks; break;
            case 0x1005: i = 3;*callbacks = ObjDict_Index1005_callbacks; break;
            case 0x1008: i = 4;break;
            case 0x1009: i = 5;break;
            case 0x100A: i = 6;break;
            case 0x1010: i = 7;break;
            case 0x1011: i = 8;break;
                ……
}
```

- 存取函数

由getODentry（）和setODentry（）函数来实现，思路如下：

– 通过扫描函数定位到指定对象的入口；

– 检查对应对象的访问权限，若越权返回错误码；

– 若权限正确，允许访问，根据调用的函数，将指定内容copy进指定对象，或将指定对象里的内容copy到目标地址（注意检查是否越界）；

– 返回。

3、报文对象

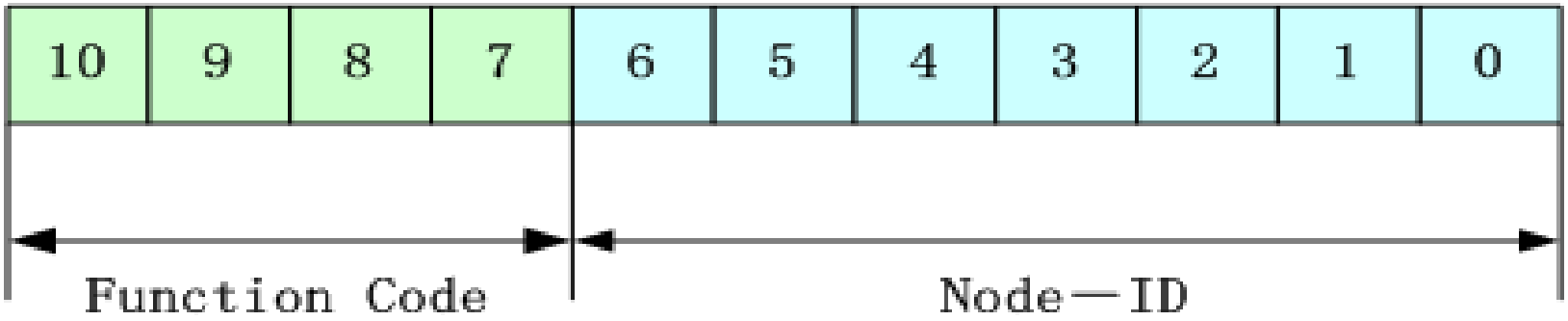
- 报文格式

  基于CAN总线的报文格式

  11位标识符（或29位）+1-8byte数据

- 标识符格式

  Node-ID由系统集成商定义，例如通过设备上的拨码开关设置。Node-ID范围是1~127（0不允许被使用）。



CANopen协议定义了四种报文对象，分别是：

- 网络管理报文对象NMT
- 服务数据对象SDO
- 过程数据对象PDO
- 预定义报文或特殊功能对象

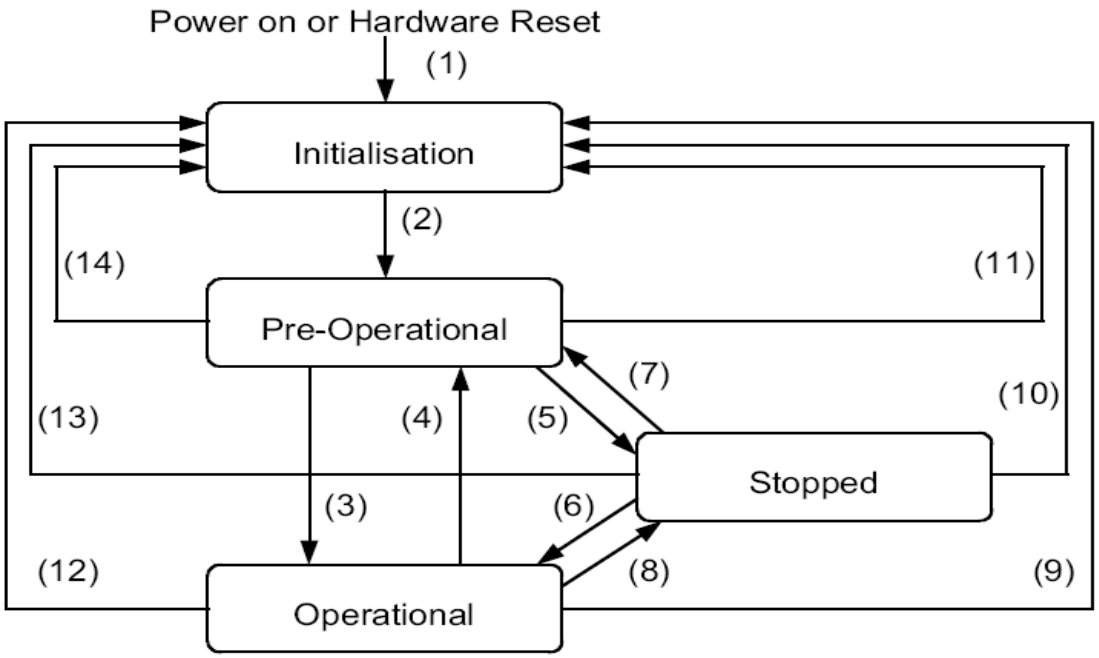
1.4 NMT

管理报文（Network Management）主要负责层管理、网络管理和ID分配服务，例如，初始化、配置和网络管理（其中包括节点保护）。

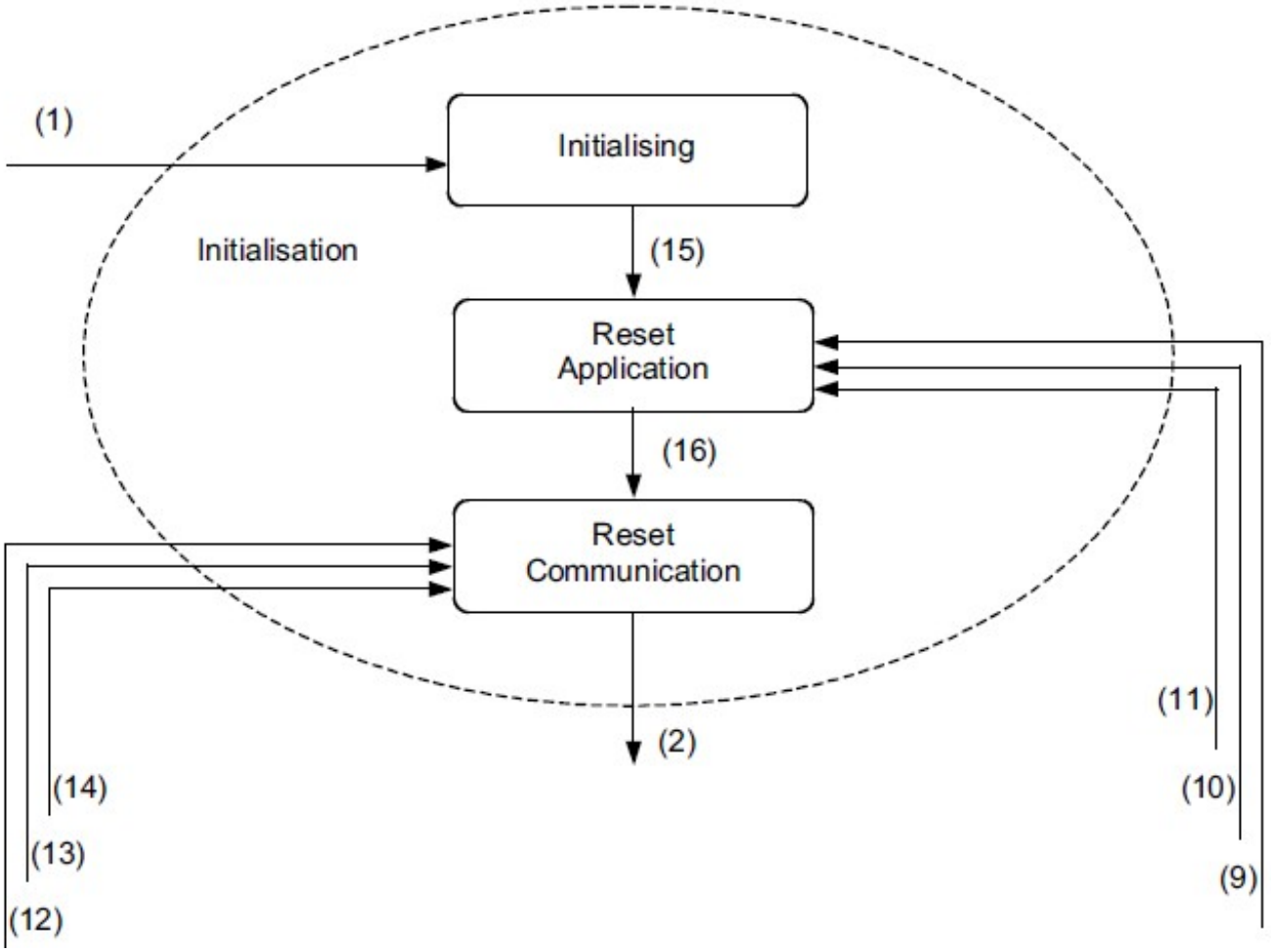
网络管理中，同一个网络中只允许有一个主节点、一个或多个从节点，并遵循主从模式。

NMT——状态机

从节点内必须实现一个状态机，其状态转换由主节点控制，不同的状态对应不同的报文对象可用性，目的是使主节点可以控制从节点在总线上的通信行为。



(1)	At Power on the initialisation state is entered autonomously
(2)	Initialisation finished - enter PRE-OPERATIONAL automatically
(3),(6)	Start_Remote_Node indication
(4),(7)	Enter_PRE-OPERATIONAL_State indication
(5),(8)	Stop_Remote_Node indication
(9),(10),(11)	Reset_Node indication
(12),(13),(14)	Reset_Communication indication



- Initialisation：从节点上电后自动进入该状态，主要完成节点的硬件初始化，通信参数和应用参数的重置，成功后自动进入Pre-Operational状态。

- Pre-Operational：在此状态中，从节点SDO被允许，主节点通过SDO完成从节点通信参数的设置，如从节点的PDO参数映射设置。主节点发送start_remote_node命令使从节点进入Operational状态。
- Operational：在此状态中，从节点完成自己的正常工作，采集数据，控制设备，与主节点通信，所有报文对象类型都被允许。
- Stopped：当从节点发生错误时，或主节点检测到从节点出错了，通过发送stop_remote_node命令使从节点进入此状态，除了接受NMT报文，其他报文对象都被禁止，以此消除对总线的不良影响。若主节点检测到错误节点恢复正常，仍可通过NMT报文让其正常运行。

只有NMT-Master节点能够传送NMT 状态控制报文（通常都是主节点）。所有从设备必须支持NMT模块控制服务。NMT 状态控制消息不需要应答。NMT消息格式如右图：第一个字节表示从节点收到命令后需要进入的状态，

NMT-Master ➔ NMT-Slave(s)

COB-ID	Byte 0	Byte 1
0x000	CS	Node-ID

命令字	NMT 服务
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-operational State
129	Reset Node
130	Reset Communication

第二个字节是接收命令的从节点ID，若为0，此时所有节点都接收该命令

- 实现

➢ 定义一个状态数组

— 包含所有的报文类型

➢ 状态改变函数

— 原子操作

— 状态报文可用性设置

— 状态转换

```
typedef struct
{
   INTEGER8 csBoot_Up;
   INTEGER8 csSDO;
   INTEGER8 csEmergency;
   INTEGER8 csSYNC;
   INTEGER8 csHeartbeat;
   INTEGER8 csPDO;
   INTEGER8 csLSS;
} s_state_communication;
```

```
case Pre_operational:
{

  s_state_communication newCommunicationState = {0, 1, 1, 1, 1, 0, 1};
  d->nodeState = Pre_operational;
  switchCommunicationState(d, &newCommunicationState);
  if (!(*(d->iam_a_slave)))
  {
    masterSendNMTstateChange (d, 0, NMT_Reset_Node);
  }
        (*d->preOperational)(d);
}
break;
```

```
void switchCommunicationState(CO_Data* d, s_state_communication *newCommunicationState)
{
#ifdef CO_ENABLE_LSS
  StartOrStop(csLSS,  startLSS(d),  stopLSS(d))
#endif
  StartOrStop(csSDO,  None,   resetSDO(d))
  StartOrStop(csSYNC, startSYNC(d),   stopSYNC(d))
  StartOrStop(csHeartbeat,  heartbeatInit(d), heartbeatStop(d))
  StartOrStop(csEmergency,  emergencyInit(d), emergencyStop(d))
  StartOrStop(csPDO,  PDOInit(d), PDOStop(d))
  StartOrStop(csBoot_Up,  None, slaveSendBootUp(d))
}
```

- 实现（续）

状态转换最终还需要报文的解析来实现报文可用性的控制。

```
canDispatch()
{
    UNS16 cob_id = UNS16_LE(m->cob_id);
    switch(cob_id >> 7)
    {
        检查COB_ID对应的报文对象在目前节点状态下是否可用；
        若可用，则进入相应的报文处理函数；  不可用则跳出；
    }
}
```

- NMT节点保护

通过节点保护服务，NMT主节点可以检查每个节点的当前状态，当这些节点没有数据传送时这种服务尤其有意义。

数据部分包括一个触发位（bit7），触发位必须在每次节点保护应答中交替置“0”或者“1”。触发位在第一次节点保护请求时置为“0”。位0到位6（bits0～6）表示节点状态，可为下表中的数值。

NMT-Master → NMT-Slave

COB-ID
0x700+Node_ID

NMT-Master ← NMT-Slave

COB-ID	Byte0
0x700 + Node_ID	Bit 7 : toggle Bit6-0 : 状态

Value	状态
0	Initialising
1	Disconnected *
2	Connecting *
3	Preparing *
4	Stopped
5	Operational
127	Pre-operational

- NMT心跳报文

从节点可被配置为产生周期性的被称作心跳报文的报文。

当一个Heartbeat节点启动后它的Bootup报文是其第一个心跳报文。

Heartbeat消费者通常是主节点，它为每个Heartbeat节点设定一个超时值，当超时发生时采取相应动作。

一个节点不能够同时支持Node Guarding和Heartbeat协议！

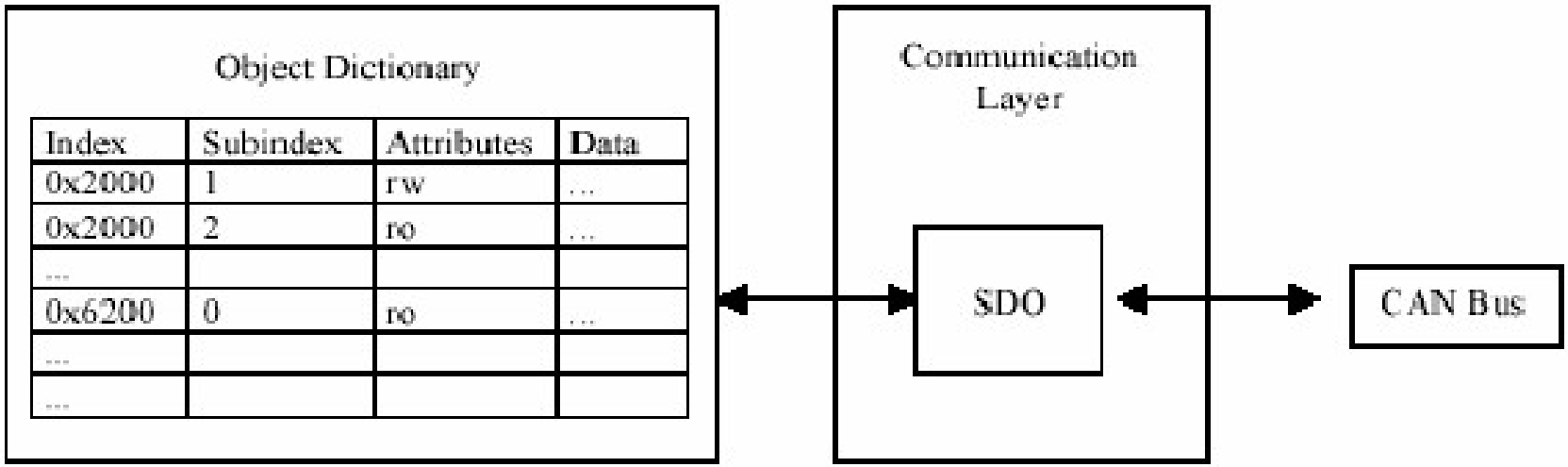
状态	意义
0	Boot-up
4	Stopped
5	Operational
127	Pre-operational

Heartbeat Producer → Consumer(s)

COB-ID	Byte 0
0x700 + Node_ID	状态

1.5 SDO

SDO是服务数据对象接口(Service Data Object)的缩写，顾名思义提供服务数据的访问接口，所谓服务数据指一些实时性要求不高的数据，一般是指节点配置参数，因此，SDO一般用来配置和获得节点的配置参数，充当OD对外的接口，其优先级只比心跳(Heartbeat)高。



SDO基于CS模式，所有报文都需要确认。通常从节点作为SDO服务器，主节点作为客户端。客户端通过索引和子索引，访问服务器上的任意对象字典，SDO的上传与下载，是从server的角度去理解的，

上传：client对server的OD进行读操作；

下载：client对server的OD进行写操作。

- 传送机制：
  - 加速传送：最多传送4Byte数据
  - 分段传送：传送数据大于4Byte
- 报文基本结构：

Client → Server / Server → Client

Byte 0	Byte 1-2	Byte 3	Byte 4-7
SDO Command Specifier	对象索引	对象子索引	**

Client → Server / Server → Client

Byte 0	Byte 1-70
SDO 命令字	最大 7 字节数据（segmented transfer）

- SDO 命令字包含如下信息：
  - 下载/上传（Download / Upload）
  - 请求/应答（Request /Response）
  - 分段/加速传送（Segmented / Expedited ）
  - CAN帧数据字节长度
  - 用于后续每个分段的交替清零和置位的触发位

SDO中实现了5个请求/应答协议：启动域下载，域分段下载，启动域上传，域分段上传和域传送中止。

命令字细节

启动域下载（Initiate Domain Download）								
Bit	7	6	5	4	3	2	1	0
Client➔	0	0	1	-	n		e	s
➔Server	0	1	1	-	-	-	-	-

说明：

- n： 如果 e=1，且 s=1，则有效，否则为 0；表示数据部分中无意义数据的字节数（字节 8—n 到 7 数据无意义）。
- e： 0= 正常传送，1= 加速传送。
- s： 是否指明数据长度，0= 数据长度未指明，1= 数据长度指明。
- e=0， s=0： 由 CiA 保留。
- e=0， s=1： 数据字节为字节计数器，byte 4 是数据低位部分（LSB），byte 7 是数据高位部分（MSB）。
- e=1： 数据字节为将要下载（download）的数据。

域分段下载（Download Domain Segment）								
Bit	7	6	5	4	3	2	1	0
Client➔	0	0	0	t	n			c
➔Server	0	0	1	t	-	-	-	-

说明：

- n： 无意义的数据字节数。如果没有指明段长度，则为 0。
- c： 0= 有后续分段需要 download，1= 最后一个段。
- t： 触发位，后续每个分段交替清零和置位（第一次传送为 0，等效于 request/response）。

命令字细节（续）

启动域上传（Initiate Domain Upload）								
Bit	7	6	5	4	3	2	1	0
Client→	0	1	0	-	-	-	-	-
←Server	0	1	0	-	n		e	s

说明：n，e，s： 与启动域下载相同。

域分段上传（Upload Domain Segment）								
Bit	7	6	5	4	3	2	1	0
Client→	0	1	1	t	-	-	-	-
←Server	0	0	0	t	n			c

说明：n，c，t： 与域分段下载相同。

SDO 客户或服务器通过发出如下格式的报文来中止 SDO 传送：

域传送中止（Abort Domain Transfer）								
Bit	7	6	5	4	3	2	1	0
C→/←S	1	0	0	-	-	-	-	-

- 实现方案——SDO线程

借鉴操作系统多线程机制的一套SDO服务处理方法，每个SDO线程表示的是与网络上其他节点建立的一个SDO链接，相当于在两个节点间建立一个SDO通讯路径，其中每个节点都拥有一个对这个路径的描述，在通讯过程中双方通过这个路径来交流，SDO通讯结束则释放该线程，每个节点可以同时与多个节点建立多个这样的路径且互不影响，就如同操作系统的多线程机制一样。

一次SDO访问是这样完成的：

- SDO发起节点(client)收集足够的信息，建立一个SDO线程，将收集的信息以特定格式放入参数表，形成一帧SDO请求，发送出去；
- server收到这帧请求，为它建立一个SDO线程，将这帧请求解析以获得足够信息来初始化这个SDO线程对应的参数表；
- 程序按照参数表的描述去执行server的功能收集数据，然后将收集到的数据形成SDO应答帧返回给client；
- 如果此次请求结束则释放该线程，否则等待下一帧请求到来；client在收到应答后判断该请求是否还有后续请求，无则释放此SDO线程；
- 如果在这个过程中，server 和client所在节点有收到其他节点的SDO请求或需要通过SDO请求其它节点，则新建SDO线程来实现，跟上面的步骤一样。

1.6 PDO

PDO(Process Data Object)被用来传输实时数据，数据从一个生产者传到一个或多个消费者，采用无确认的方式，数据长度被限制为1~8字节。PDO通讯没有协议规定。PDO数据内容只由它的映射参数对象定义，假定生产者和消费者知道这个PDO的数据内容。

- PDO通讯参数：包含哪个COB-ID将被PDO使用，传输类型，禁止时间和定时器周期。

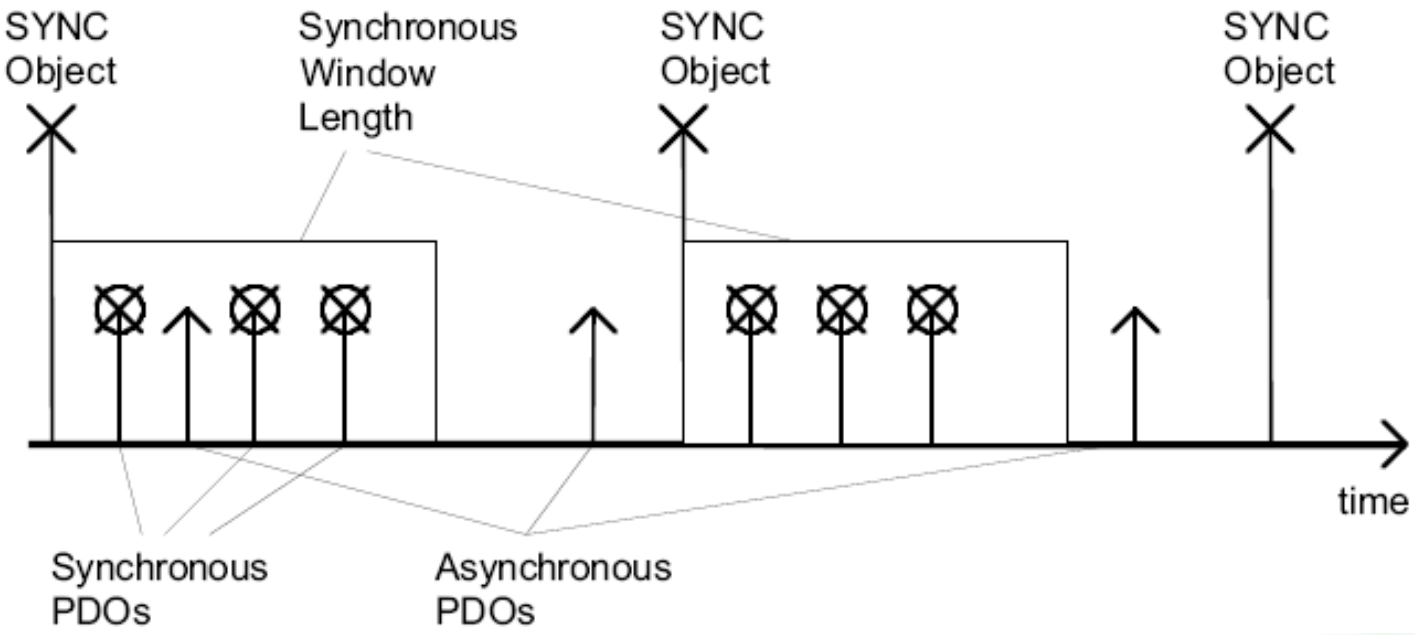
Index	Sub-Index	Description	Data Type
1XXXh	0h	number of entries	Unsigned8
	1h	COB-ID	Unsigned32
	2h	transmission type	Unsigned8
	3h	inhibit time	Unsigned16
	4h	reserved	Unsigned8
	5h	event timer	Unsigned16

- PDO通讯参数
  - PDO可以指定禁止时间，避免由于高优先级信息的数据量太大，始终占据总线，而使其它优先级较低的数据无力竞争总线的问题。禁止时间单位100us。
  - PDO可以指定一个事件定时周期，当超过定时时间后，一个PDO传输可以被触发（不需要触发位）。事件定时周期由16位无符号整数定义，单位1ms。

- PDO传输模式
  - 异步触发模式
  - 远程索取模式
  - 同步触发模式
    - 同步周期模式
    - 同步非周期模式
- PDO的同步传输模式

同步传输（通过接收SYNC对象实现同步），同步传输又可分为非周期和周期传输。非周期传输是由远程帧预触发或者由设备子协议中规定的对象特定事件预触发传送。周期传输则是通过接收同步对象（SYNC）来实现。

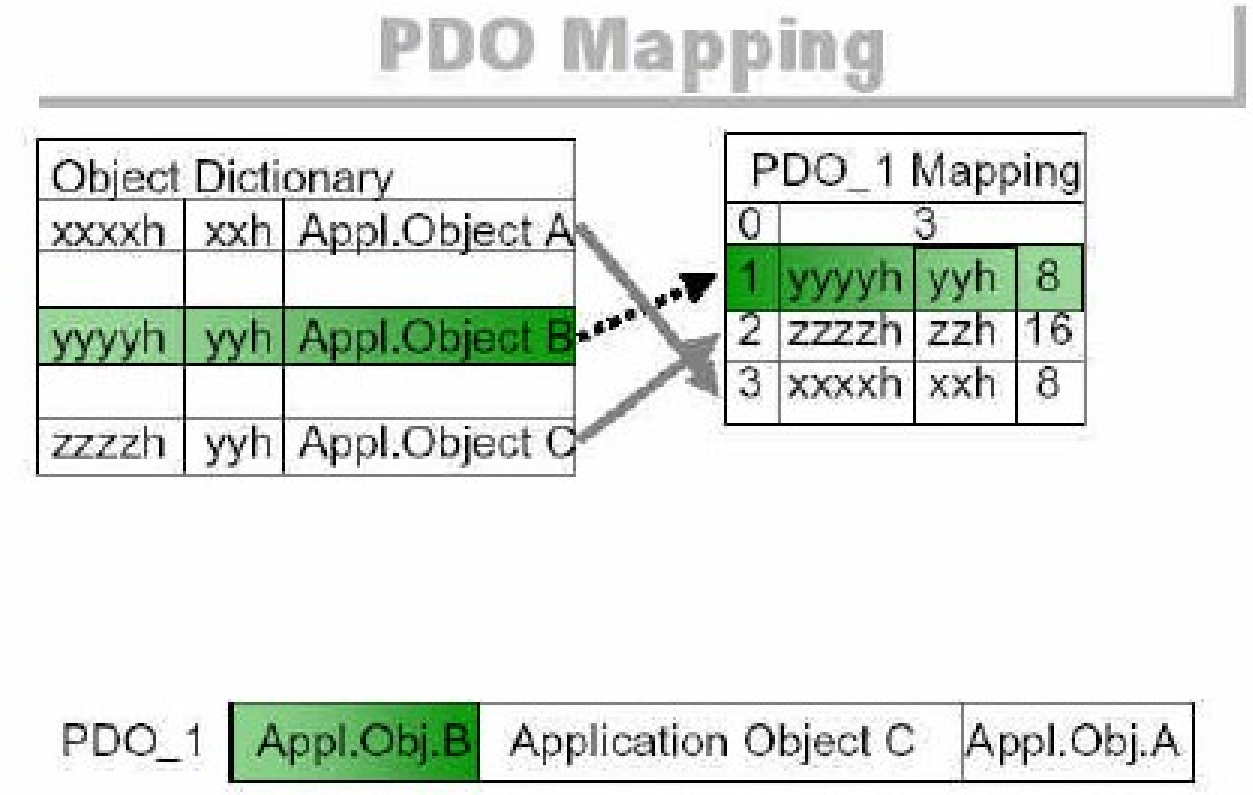
每个SYNC后，有一段同步时间窗口，同步PDO在窗口内发送。可设置同步PDO的发送速率，如传输类型为0，表示PDO仅在有事件发生SYNC前，在时间窗口内发送。若为N(1-240)，传送在每N个SYNC消息后触发。



- PDO的异步传输模式

其触发方式可有两种，第一种是通过发送与PDO的COB-ID相同的远程帧来触发PDO的发送（传输类型为253），第二种是由设备子协议中规定的对象特定事件来触发（例如，定时传输，数据变化传输等，传输类型为255）。

- PDO映射参数：包含一个对象字典中对象的列表，这些对象映射到PDO里，包括它们的数据长度。生产者和消费者必须知道这个映射，以解释PDO内容。



举个例子，一个PDO的映射参数对象（1A01）：

对象 0x1A01：第二个 Transmit-PDO 映射		
子索引	值	意义
0	2	2 个对象映射到 PDO 中
1	0x60000208	对象 0x6000，子索引 0x02，由 8 位组成
2	0x64010110	对象 0x6401，子索引 0x01，由 16 位组成

则这个PDO报文的内容是：

COB-ID	Byte 0	Byte 1	Byte 2
0x280+Node_ID	8 位数据量输入	16 位模拟量输入（低 8 位）	16 位模块量输入（高 8 位）

改变1A01里面的内容，就可以改变该PDO报文的数据内容。

- 实现

sendPDOevent函数：

- 用于辅助proceedPDO函数；
- 循环更新每个PDO的当前trans_type_para；
- 若当前传输类型等于原传输类型，调用buildPDO，发送PDO；
- 若传参类型为TRANS_RTR_SYNC,则buildPDO,为远程请求的应答做好准备，设置当前状态为RTR_SYNC_READY；
- 若当前传参类型为TRANS_SYNC_ACYCLIC或EVENT相关且未被禁止，则buildPDO，为事件触发做好准备；
- 若不是同步事件，且事件周期和禁止时间不为0，则设置两个定时器，在计时器到达前将PDO状态设为禁止。

- 实现（续）

proceed PDO函数：

- 接收到PDO报文，有两种类型：
- 若RTR为0，表示普通PDO报文，对从节点就是主节点发出的控制命令，需要从节点来执行。对普通PDO报文的处理：找到对应COB-ID的PDO映射参数，并把数据写入相应的对象中，等待应用来执行；
- 若RTR为1，表示远程请求，根据传输类型不同作相应处理；
- 若传输类型为TRANS_RTR，则buildPDO，发送应答PDO；
- 若传输类型为TRANS_RTR_SYNC，且当前状态为RTR_SYNC_READY，则直接发送PDO；
- 若传输类型为EVENT相关，则调用PDOEventTimerAlarm函数等待定时器到期，自动触发PDO发送。

1.7 特殊功能对象

预定义报文或特殊功能对象为CANopen设备提供特定的功能，方便CANopen主站对从站管理。在CANopen协议中，已经为特殊的功能预定义了COB-ID，其主要有以下几种特殊报文：

- 紧急事件对象（Emergency），当设备内部发生错误时触发该对象，即发送设备内部错误代码；
- 时间标记对象（Time Stamp），为各个节点提供公共的时间参考；

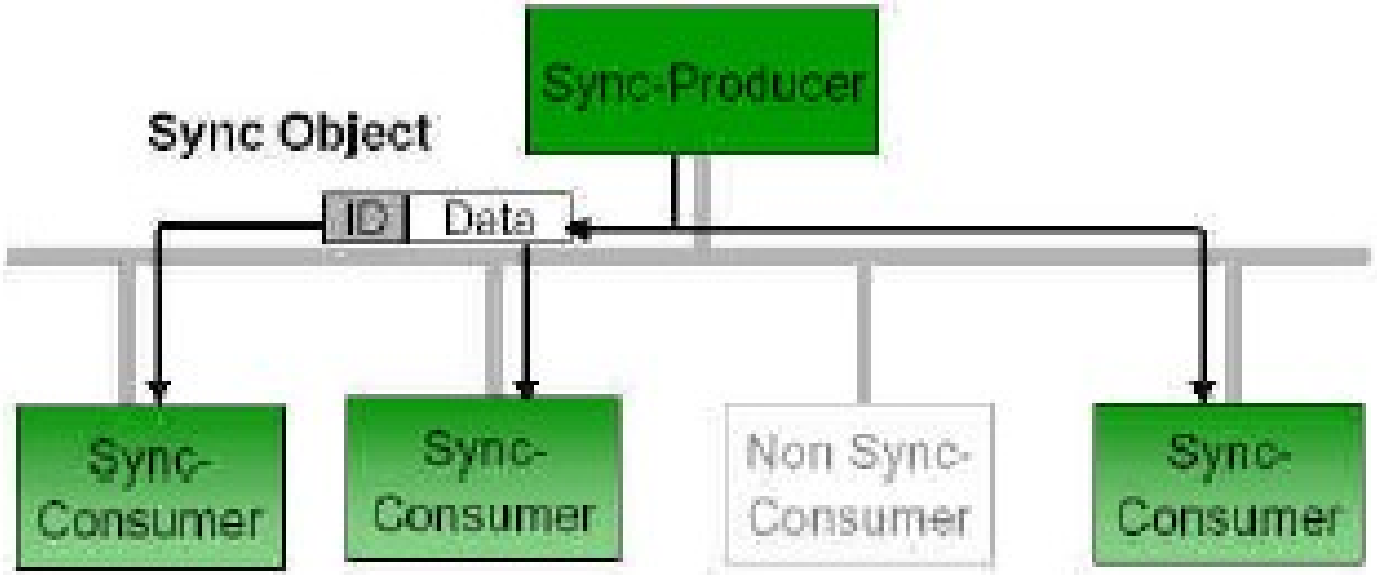
- 节点/寿命保护（Node/Life Guarding），主节点可通过节点保护方式获取从节点的状态。从节点可通过寿命保护方式获取主节点的状态；
- 启动报文对象（Boot-up），从节点初始化完成后向网络中发送该对象，并进入到Pre-Operational状态。

NMT-Master ← NMT-Slave

COB-ID	Byte 0
0x700 + Node_ID	0

特殊功能对象——SYNC

同步（SYNC），该报文对象基于生产者/消费者模式，由SYNC生产者周期性的广播，作为网络基本时钟，实现整个网络的同步传输，每个节点都以该同步报文作为同步PDO触发参数，因此该同步报文的COB-ID具有比较高的优先级以及最短的传输时间。



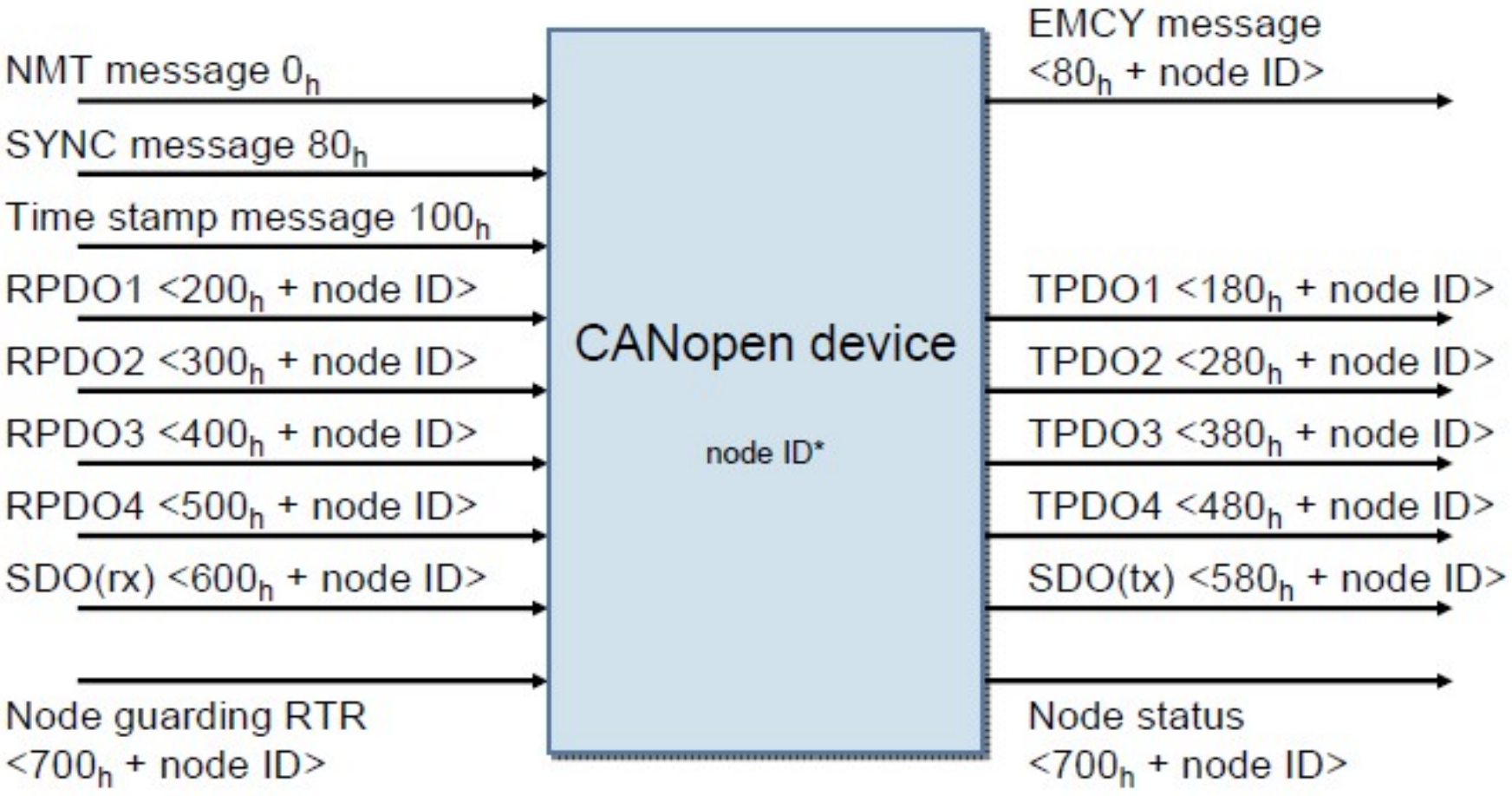
- 描述(对象字典)
  - 1005H：存放SYNC的COB-ID
  - 1006H：存放SYNC的通信周期
  - 1007H：存放SYNC时间窗口
- 实现
  - sendSYNC：由定时器中断来调用，定时值就是通信周期；
  - proceedSYNC：检查节点是否在Operation状态，调用sendPDOevent函数来处理同步PDO；

预定义连接集

由于动态分配CAN-ID将占用较多的系统资源，对于一些相对简单的CANopen网络，动态分配标识符也没有太大的必要性，为减少简单网络的组态工作量，CANopen定义了强制性的缺省标识符（CAN-ID）分配表。这些标志符在预操作状态下可用，通过动态分配还可修改他们。

预定义连接集定义了4个接收PDO（Receive－PDO），4个发送PDO（Transmit－PDO），1个SDO（占用2个CAN-ID），1个紧急对象和1个节点错误控制(Node-Error-Control)ID。也支持不需确认的NMT-Module-Control服务，SYNC和Time Stamp对象的广播。

CANopen 预定义主/从连接集的广播对象			
对象	功能码 （ID-bits 10-7）	COB-ID	通讯参数在 OD 中的索引
NMT Module Control	0000	000H	-
SYNC	0001	080H	1005H，1006H，1007H
TIME SSTAMP	0010	100H	1012H，1013H
CANopen 主/从连接集的对等对象			
对象	功能码 功能码 （ID-bits 10-7）	COB-ID	通讯参数在 OD 中的索引
紧急	0001	081H-0FFH	1024H，1015H
PDO1(发送)	0011	181H-1FFH	1800H
PDO1(接收)	0100	201H-27FH	1400H
PDO2(发送)	0101	281H-2FFH	1801H
PDO2(接收)	0110	301H-37FH	1401H
PDO3(发送)	0111	381H-3FFH	1802H
PDO3(接收)	1000	401H-47FH	1402H
PDO4(发送)	1001	481H-4FFH	1803H
PDO4(接收)	1010	501H-57FH	1403H
SDO(发送/服务器)	1011	581H-5FFH	1200H
SDO(接收/客户)	1100	601H-67FH	1200H
NMT Error Control	1110	701H-77FH	1016H-1017H


NMT message 0_h
SYNC message 80_h
Time stamp message 100_h
RPDO1 <200_h + node ID>
RPDO2 <300_h + node ID>
RPDO3 <400_h + node ID>
RPDO4 <500_h + node ID>
SDO(rx) <600_h + node ID>
Node guarding RTR
<700_h + node ID>
CANopen device
node ID*
EMCY message
<80_h + node ID>
TPDO1 <180_h + node ID>
TPDO2 <280_h + node ID>
TPDO3 <380_h + node ID>
TPDO4 <480_h + node ID>
SDO(tx) <580_h + node ID>
Node status
<700_h + node ID>
* node ID 0 is reserved

CANopen支持多种类型设备模块，不同的设备子协议(DS 4XX)，对于预定义连接集中的PDO定义是不同的。

例如CiA DS 401作为I/O模块的设备子协议，详细地规定了各个PDO的通信参数和数据映射参数。

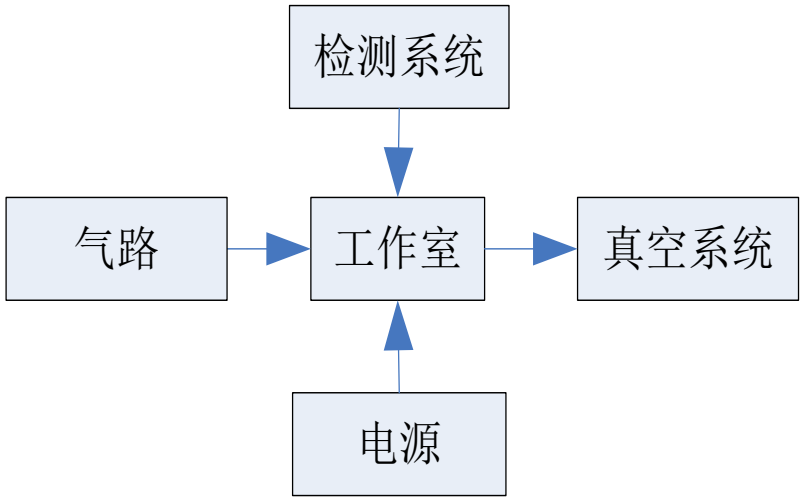
四组PDO中一组用于DI/DO，其余三组用于AI/AO，若模块只实现其中的部分功能，则其他功能部分可以不使用。

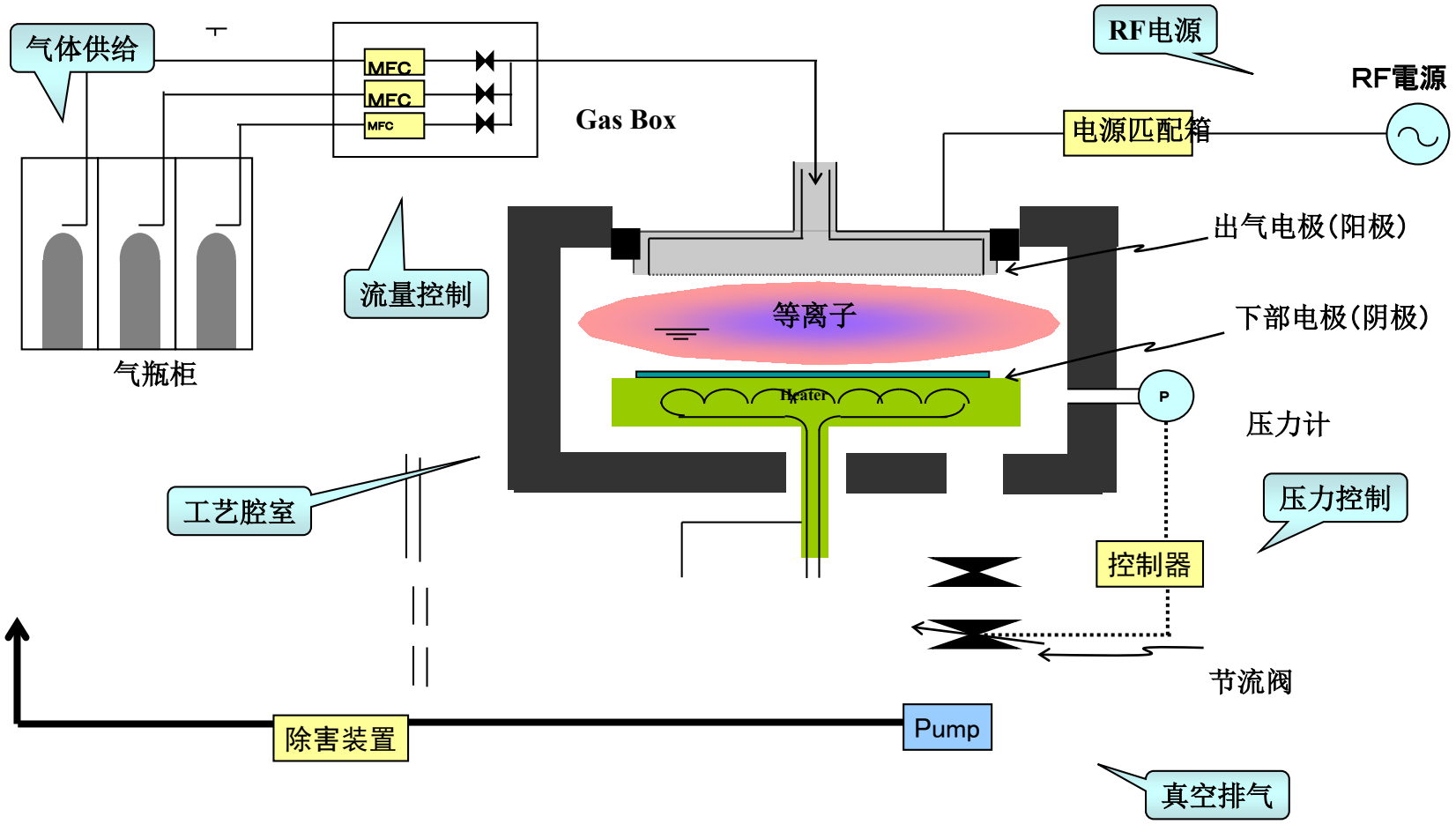
- DS 401
  - RPDO1:映射到6200H的8个8bit array (DO)
  - TPDO1:映射到6000H的8个8bit array (DI)
    - 对应最大64个DI/DO（Boolean）
  - RPDO2:映射到6411H的前4个16bit array（AO）
  - TPDO2:映射到6401H的前4个16bit array（AI）
  - RPDO3:映射到6411H的中4个16bit array（AO）
  - TPDO3:映射到6401H的中4个16bit array（AI）
  - RPDO4:映射到6411H的后4个16bit array（AO）
  - TPDO4:映射到6401H的后4个16bit array（AI）
    - Integer类型

二　CANopen协议具体应用开发

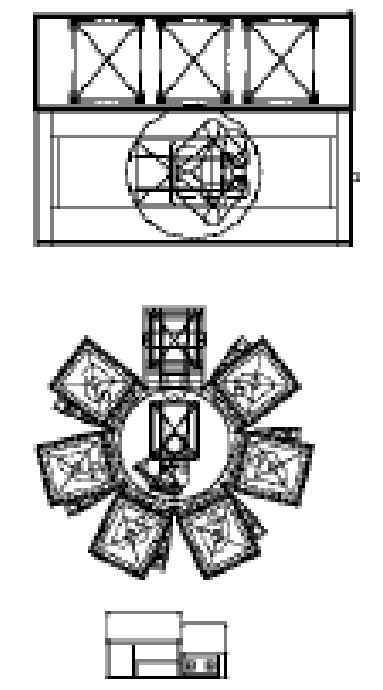
2.1 应用对象

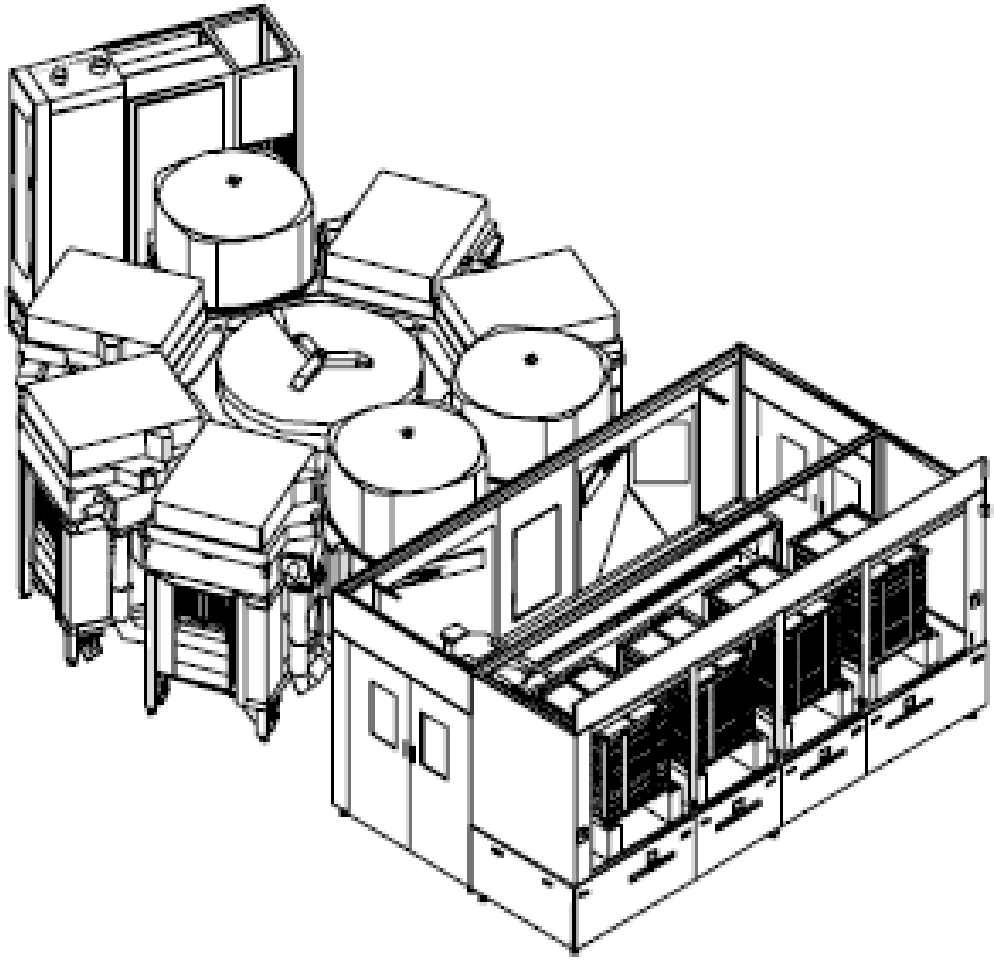
低温等离子体设备系统。





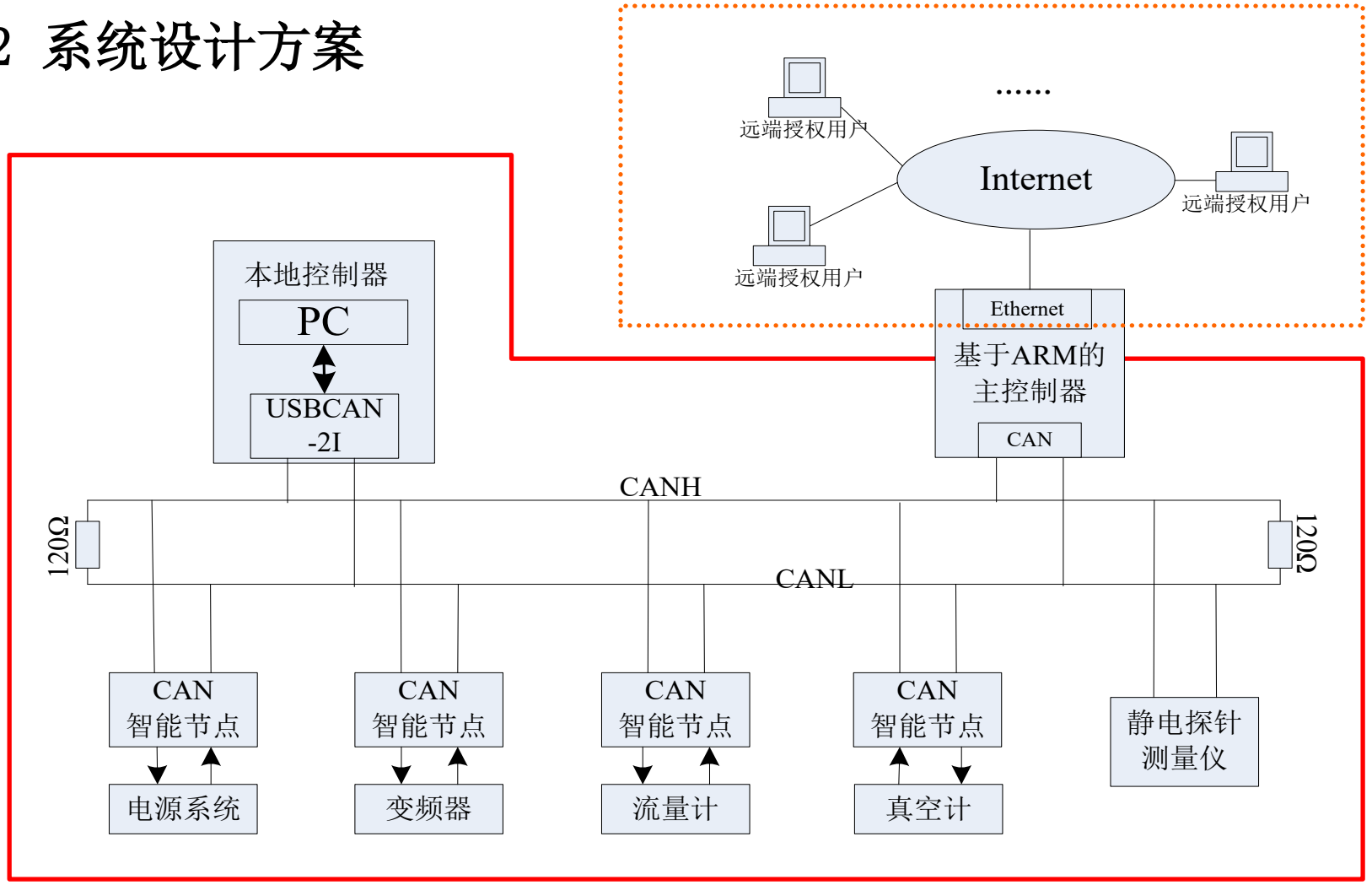
PECVD设备系统概念图



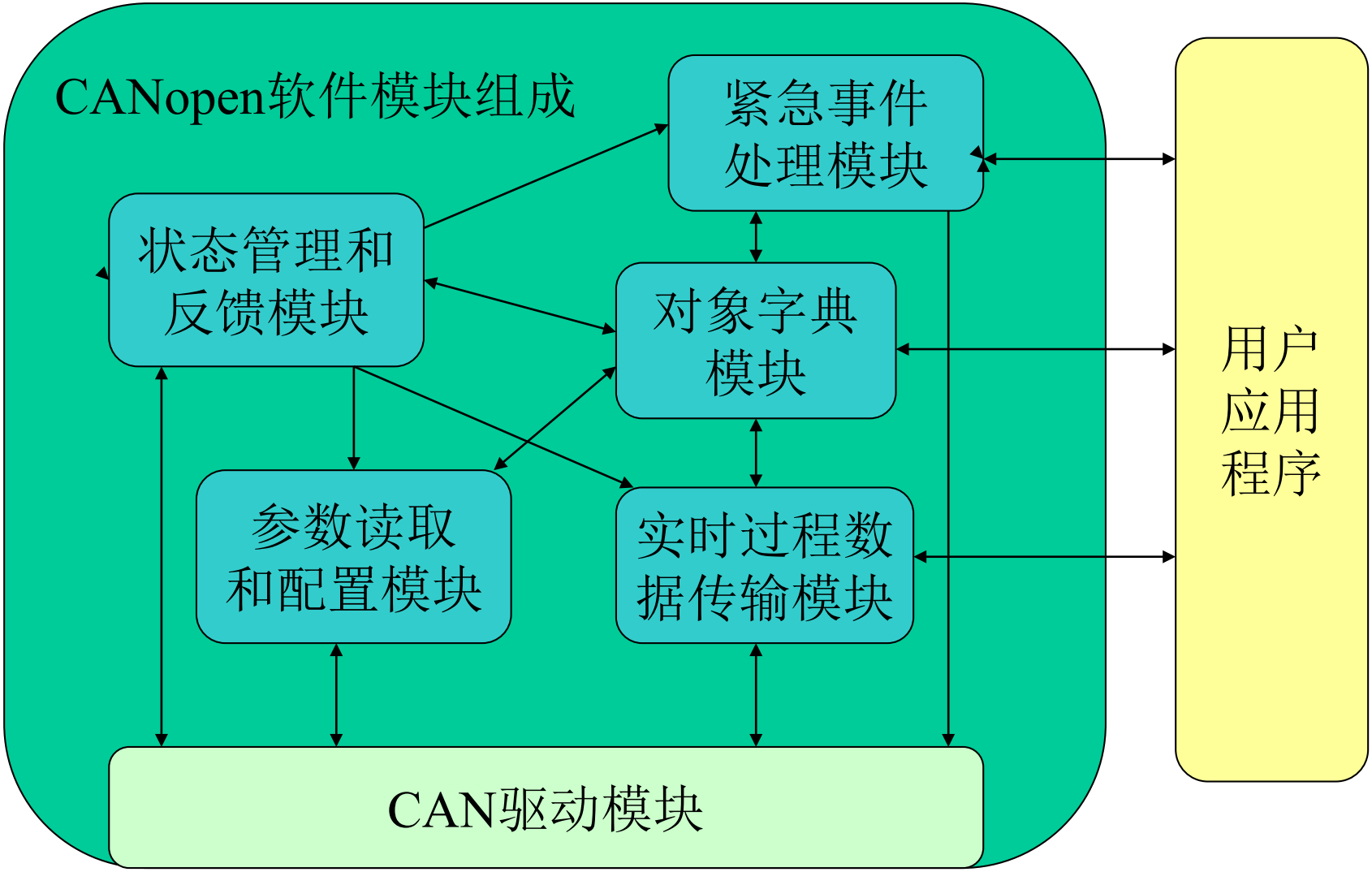


PECVD设备组成结构示意图

2.2 系统设计方案



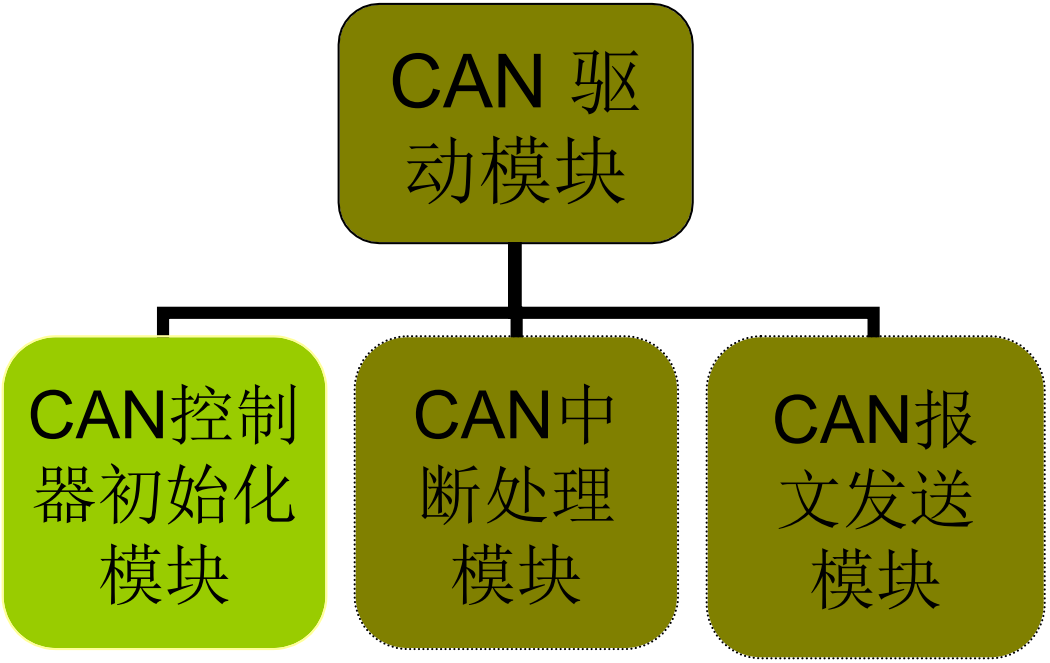
基于CAN总线的低温等离子体设备系统设计方案

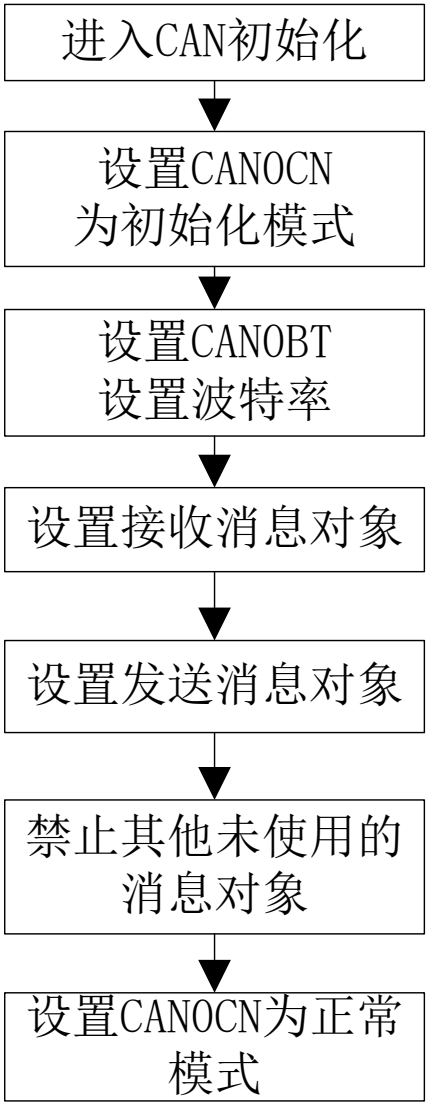

CANopen软件模块组成
紧急事件处理模块
状态管理和反馈模块
对象字典模块
参数读取和配置模块
实时过程数据传输模块
CAN驱动模块
用户应用程序

系统软件总体框架

2.3 软件功能模块设计

2.3.1 CAN驱动程序

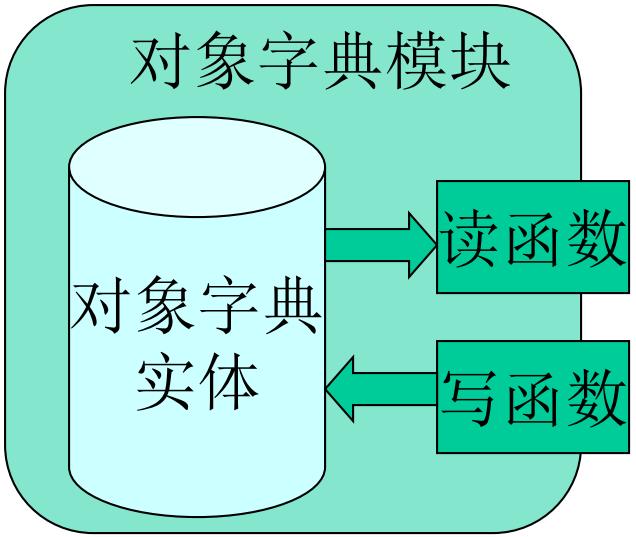




2.3.2 对象字典

对象字典是CANopen的核心概念，它存储了节点的所有信息。是一个有序的对象组，每个对象采用一个16位的索引值来寻址，同时定义了一个8位的子索引用于访问数据结构中的单个元素。

- 对象字典功能框图



- 对象字典实体结构

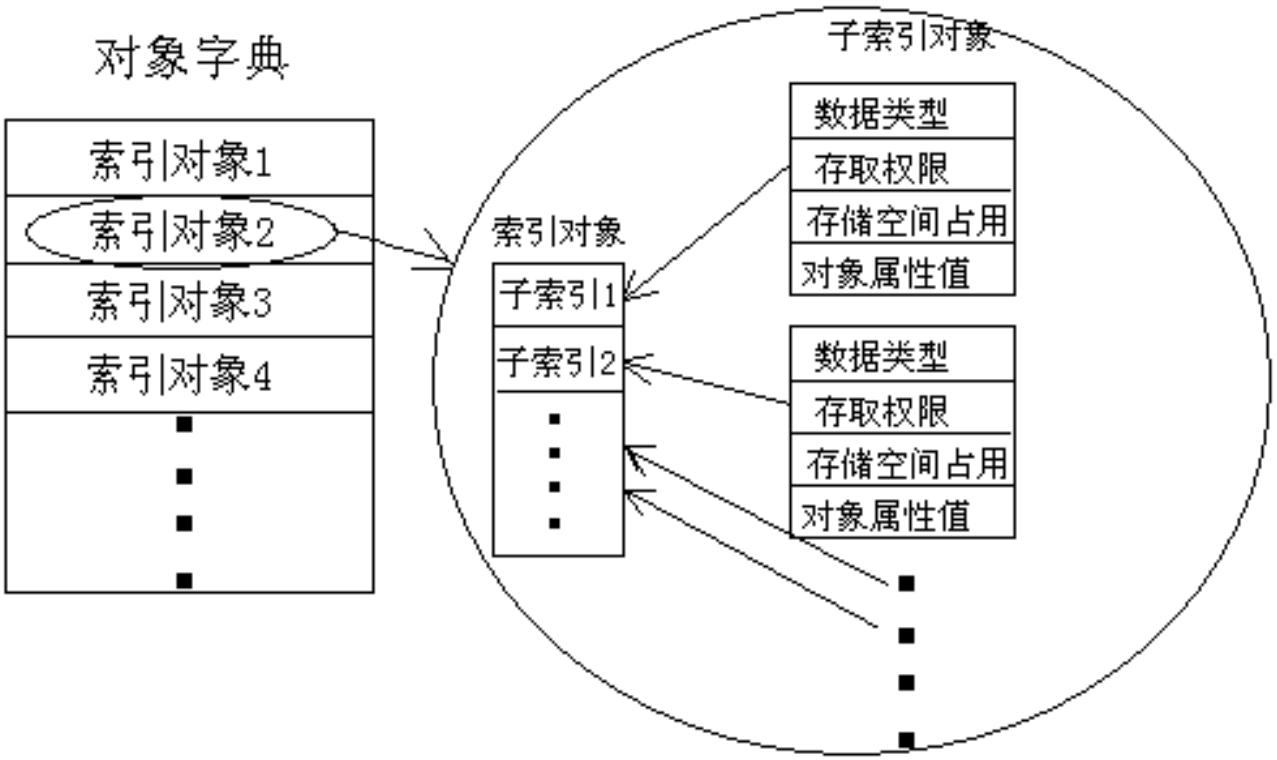
0001H-0FFFH数据类型定义区(可选)
1000H-1FFFH通信子协议区(必要)
2000H-5FFFH制造商特定协议区(可选)
6000H-9FFFH标准设备子协议区(必要) 存储电流电压输入信号、电压控制信号、启/停、手/自动….

- 实现方案

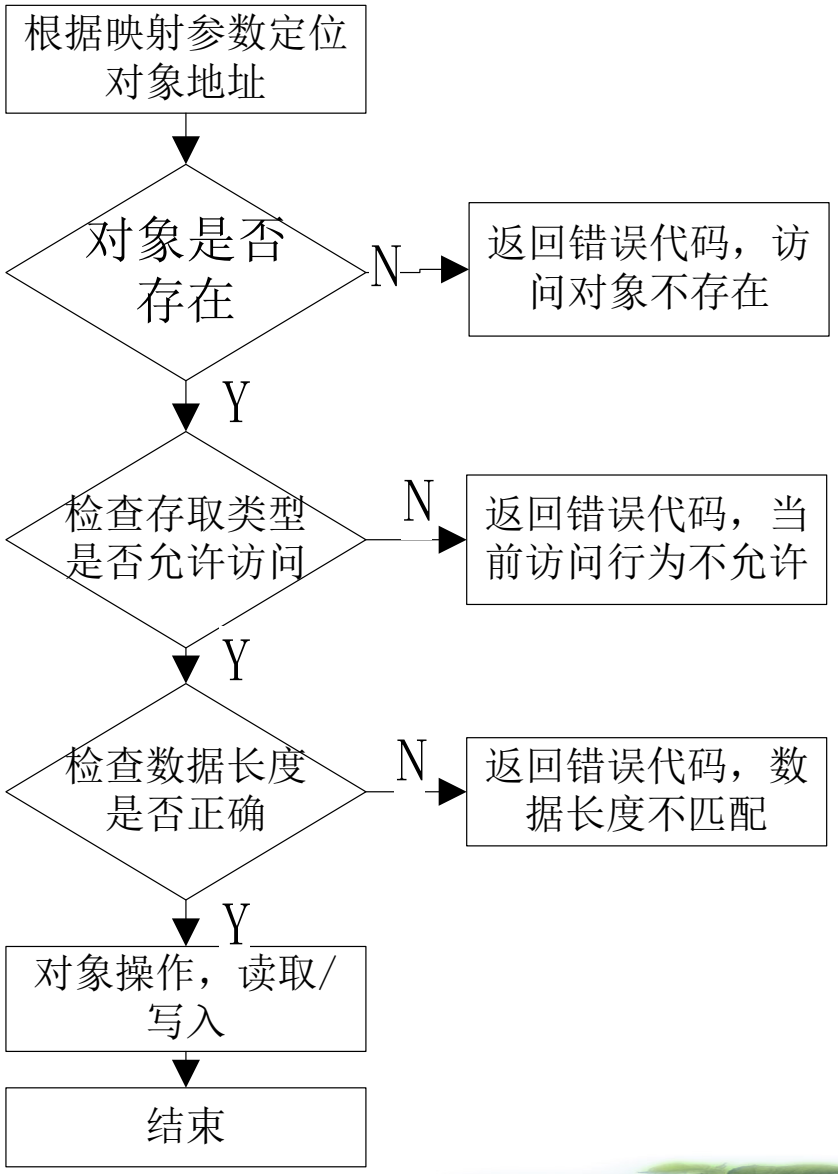
  单片机计算资源紧缺，采用静态结构体数组

  – 结构简单，易于实现

  – 访问效率高

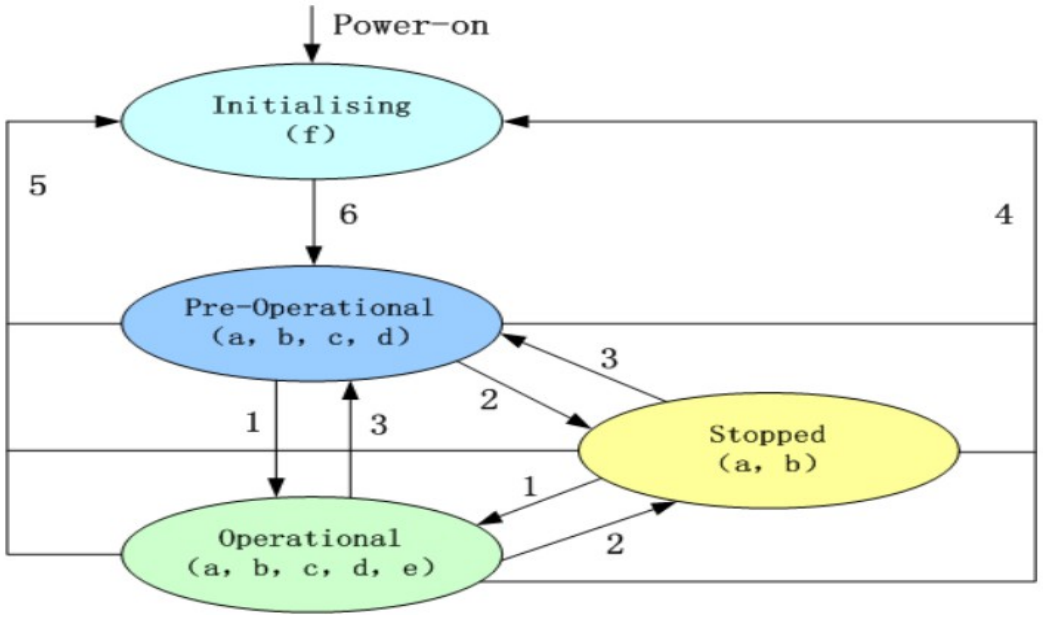


- 对象字典接口函数
  - getODentry、setODentry
  - 与对象字典配套使用
  - 访问指定对象
  - 控制访问行为，杜绝非法访问，防止数据损坏



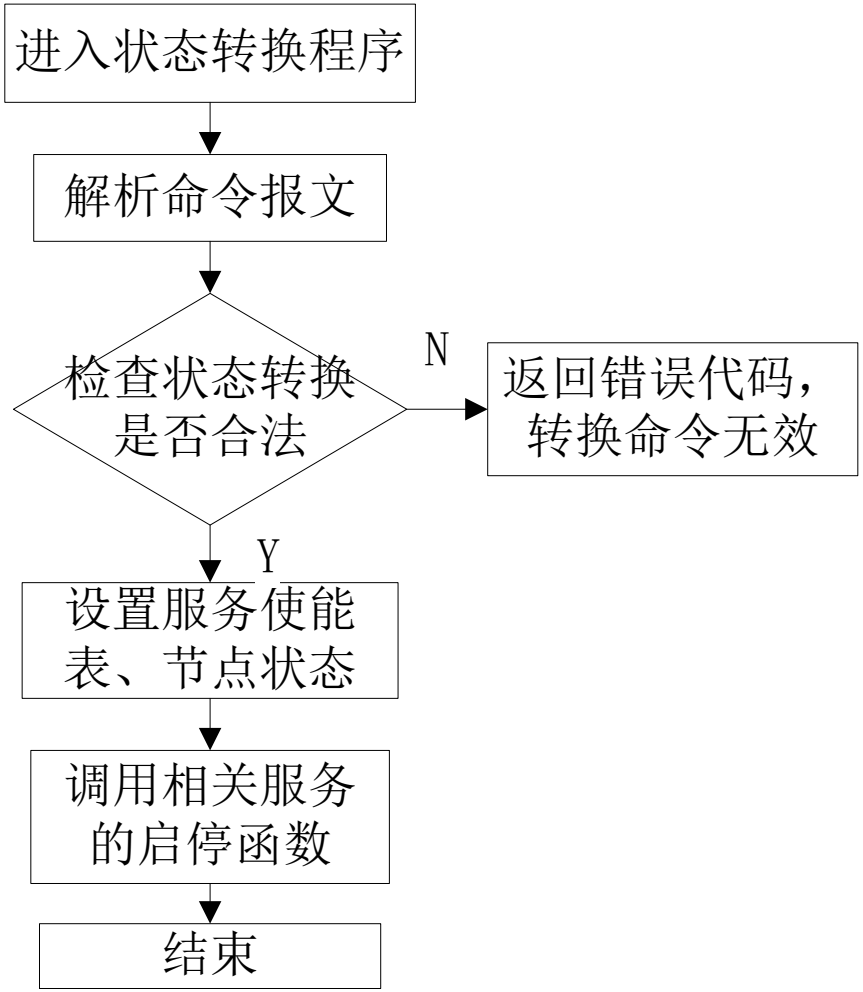
2.3.3 状态控制

在节点内部运行着一个状态机，状态转换由管理者控制，节点在不同的状态下执行不同的任务，各个功能模块的使能情况也不同。



1	启动远程节点
2	暂停远程节点
3	进入预操作状态
4	重置节点
5	重置节点通信
6	初始化完成后自动进入预操作状态

- 状态转换过程

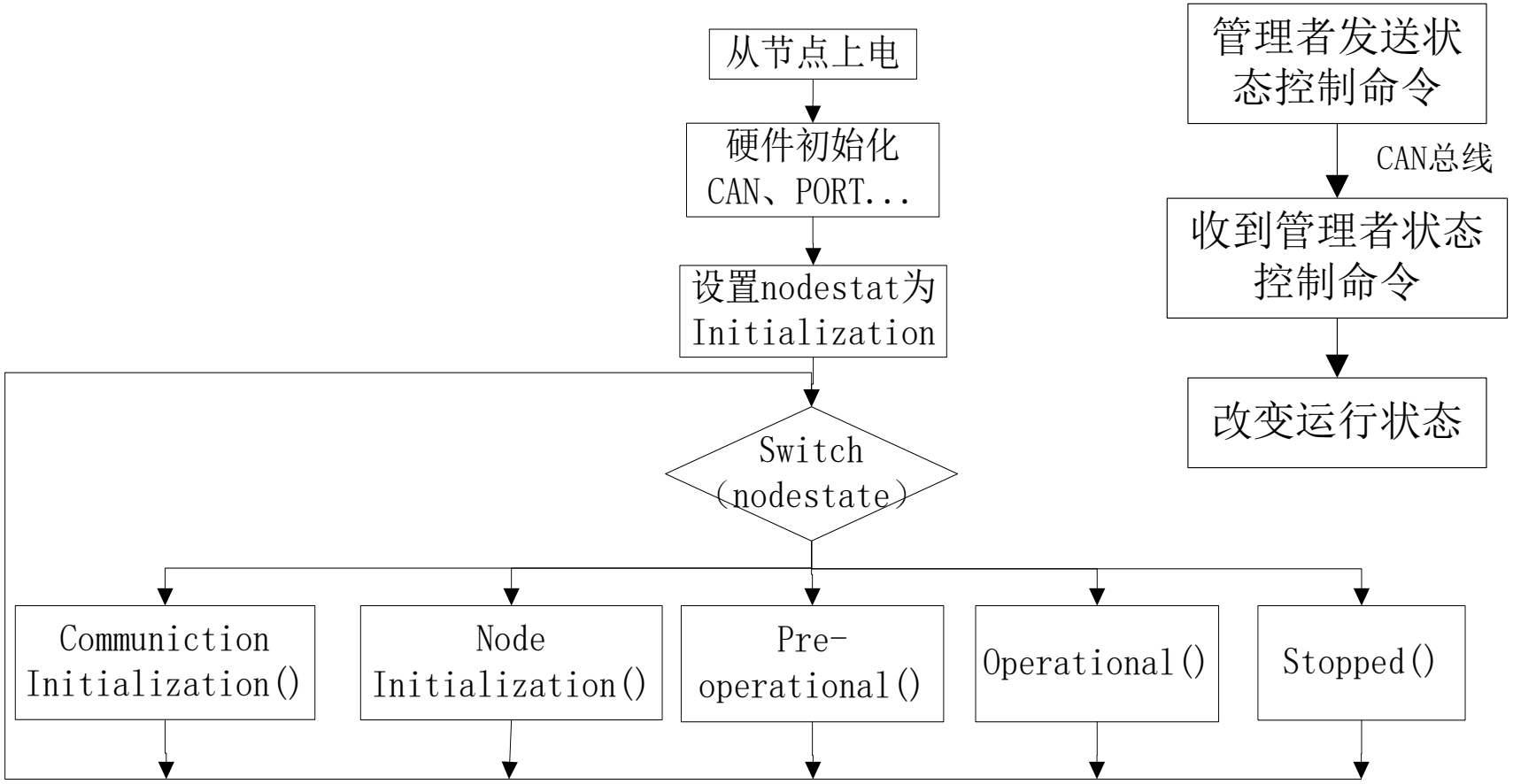


- 功能可用性控制
  - 调用相应的功能模块时检查功能使能表。
  - 只有服务被使能，调用才生效。

服务使能表

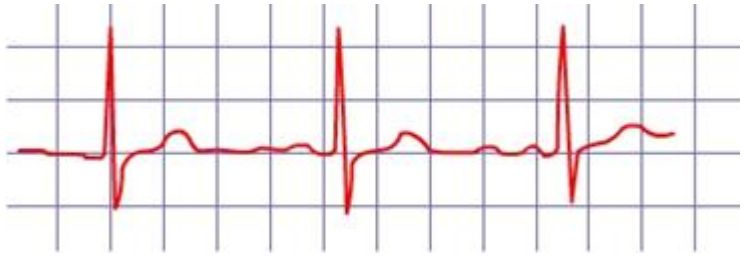
启动报文服务
紧急事件服务
心跳服务
实时数据传输服务
参数配置服务
状态控制服务

- 从节点上电后在状态机里运行，状态受管理者控制，设计从节点的软件总体流程如下：



2.3.4 状态反馈

节点可以通过一种叫做心跳的方式周期性地向主站汇报当前的状态。



- 生产者/消费者模式
- 检测节点是否在线
- 监视节点状态
- 定时器中断处理发送，定时值取自对象字典。

Heartbeat Producer ➔ Consumer(s)

COB-ID	Byte 0
0x700 + Node_ID	状态

状态	意义
0	Boot-up
4	Stopped
5	Operational
127	Pre-operational

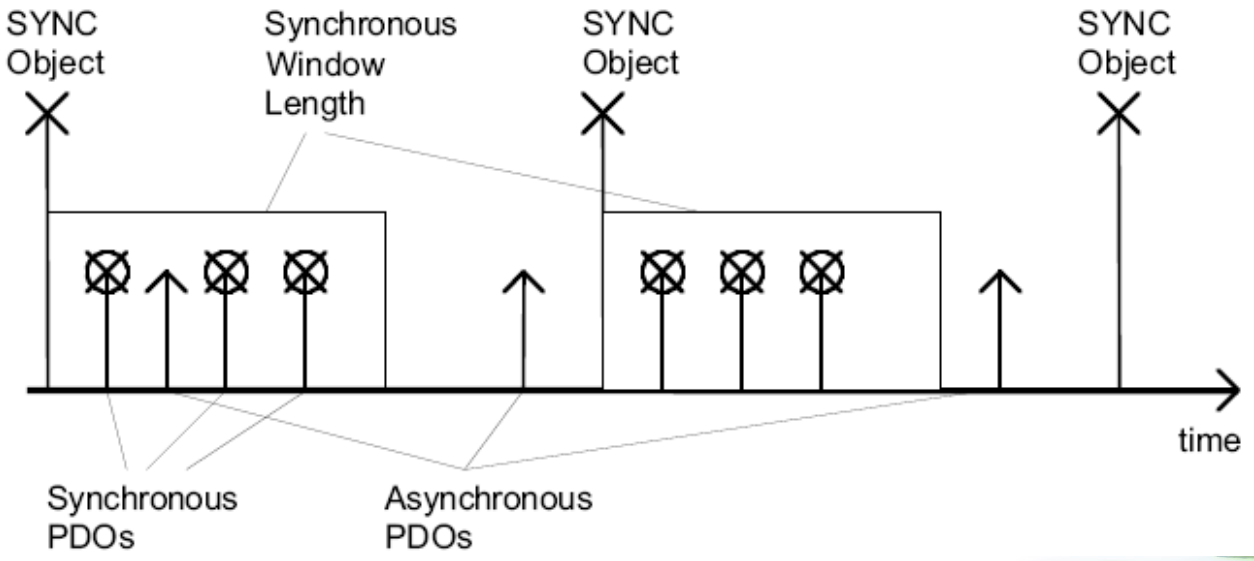
2.3.5 实时过程数据传输

- 引入CANopen的目的之一：更规范和有效地传输实时过程数据。
- 两个问题：
  - I/O数据什么时候被传输？
  - 数据如何封装？
- 利用CANopen协议中的PDO(Process Data Object)功能
  - 基于生产者/消费者模式。
  - 多种传输模式。
  - 数据长度被限制为1~8字节。
  - 数据内容由映射参数对象决定，双方都知道。

- 多种传输模式

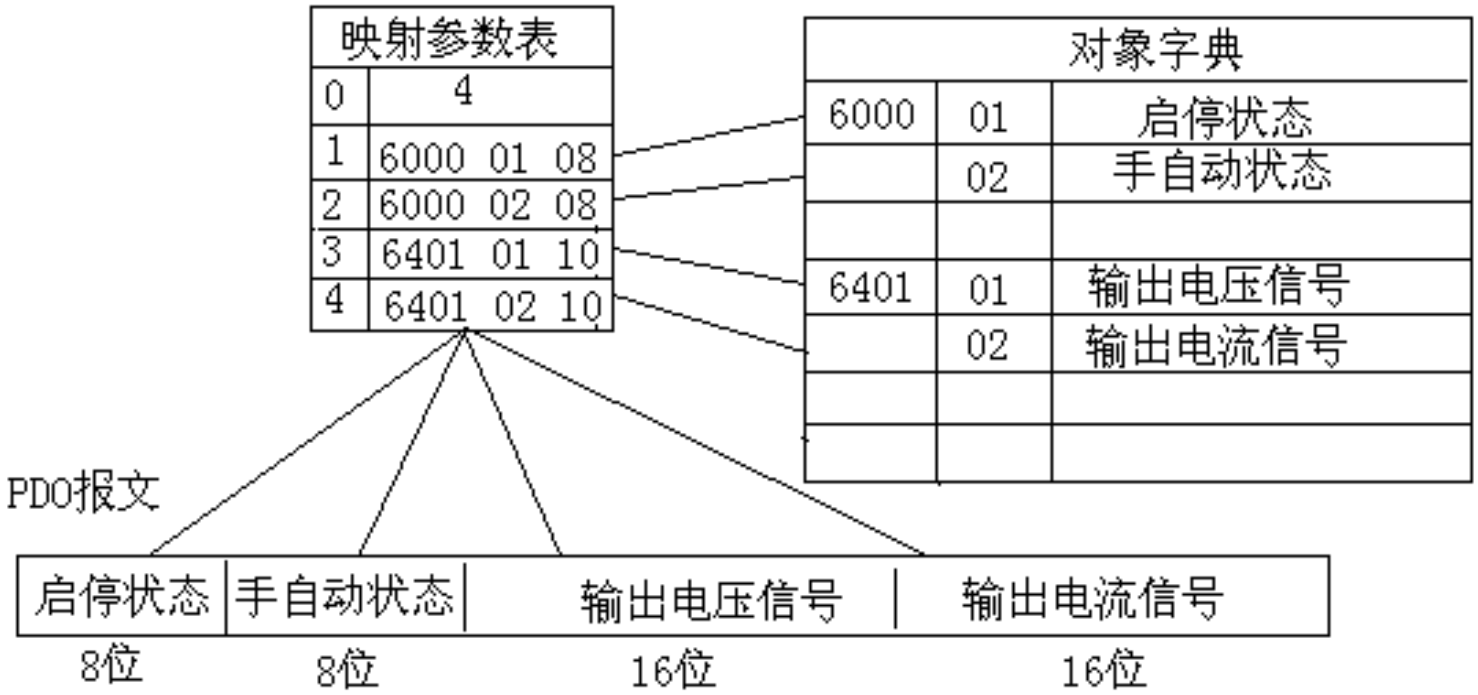
同步
- 同步周期
- 同步非周期
  - 远程帧预触发
  - 事件预触发

异步
- 远程帧触发
- 事件触发

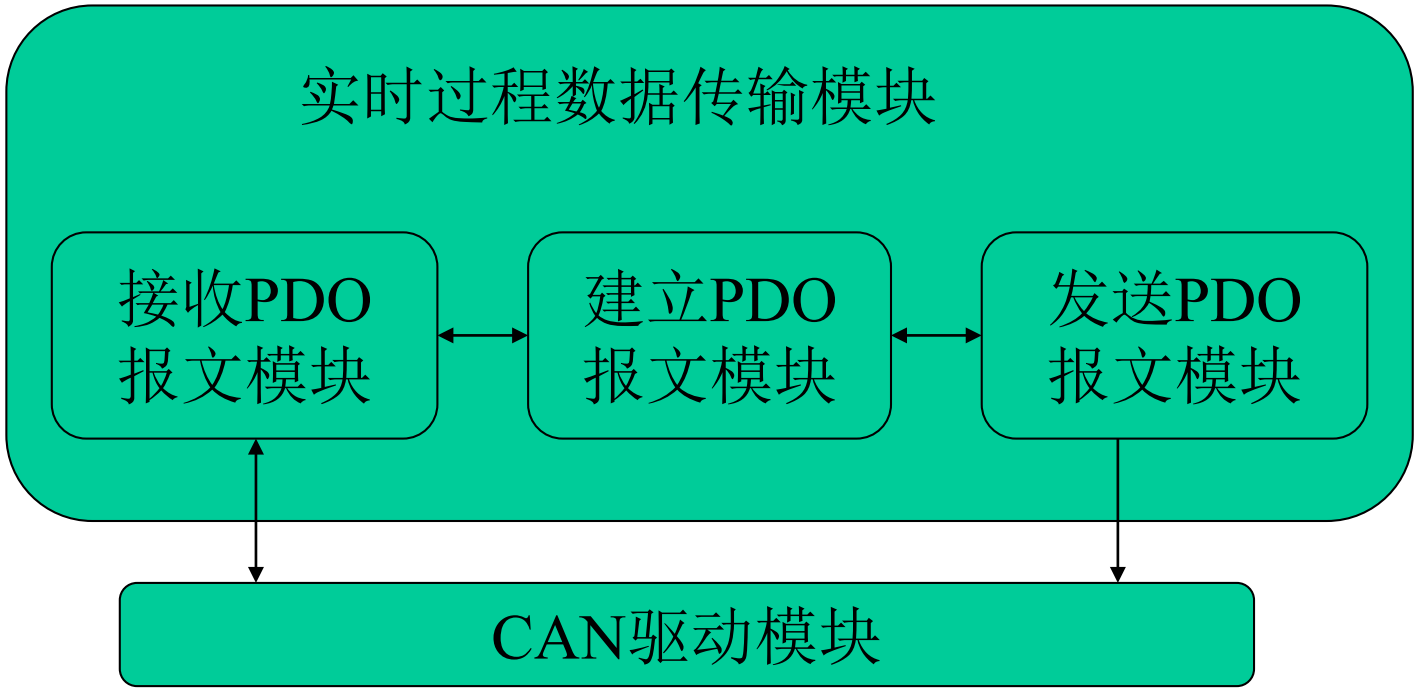


- 映射参数

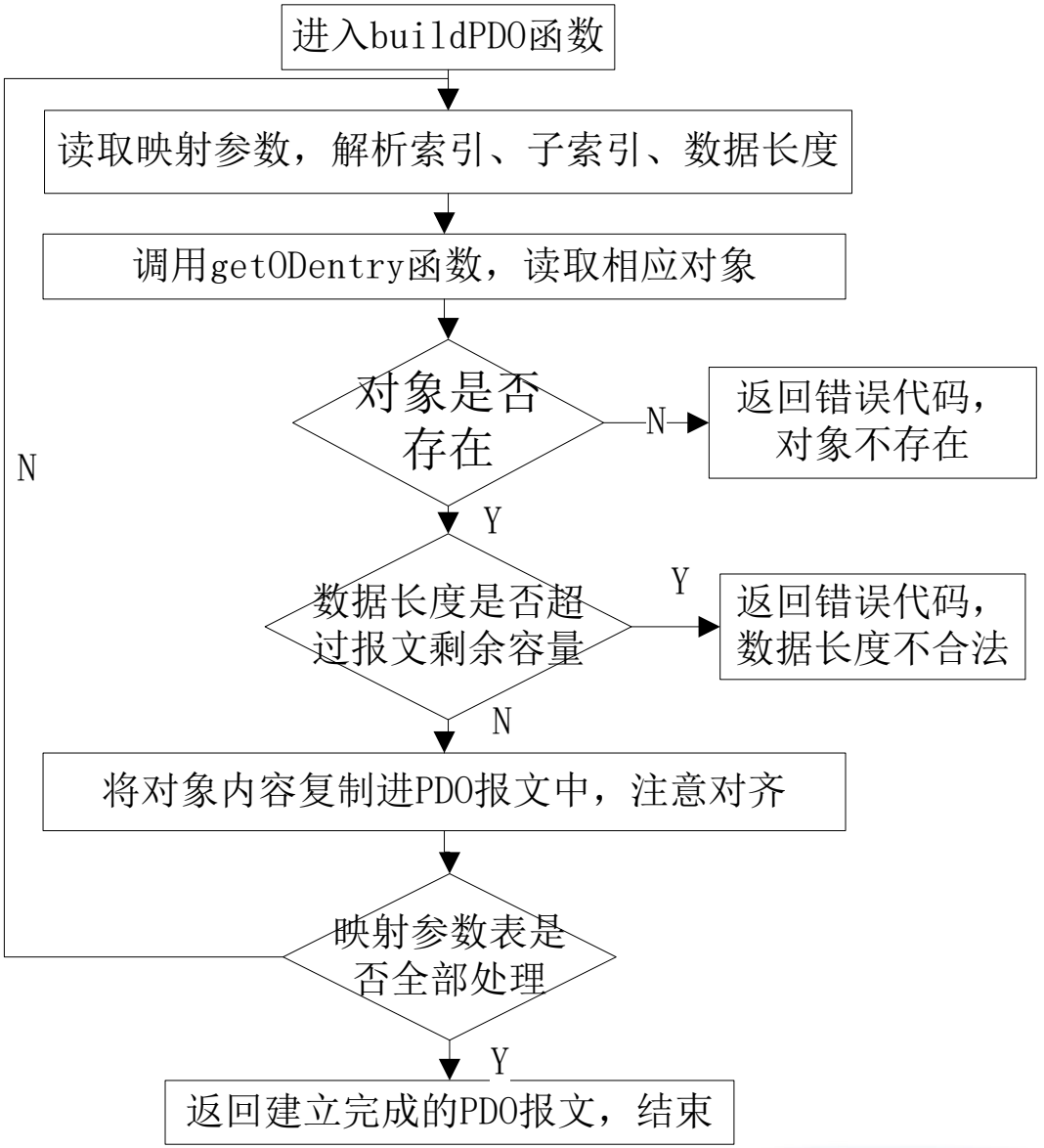
包含一个对象字典中对象的列表，这些对象映射到PDO报文里，包括对象的数据长度。生产者和消费者必须知道这个映射，才能解释PDO报文的内容。



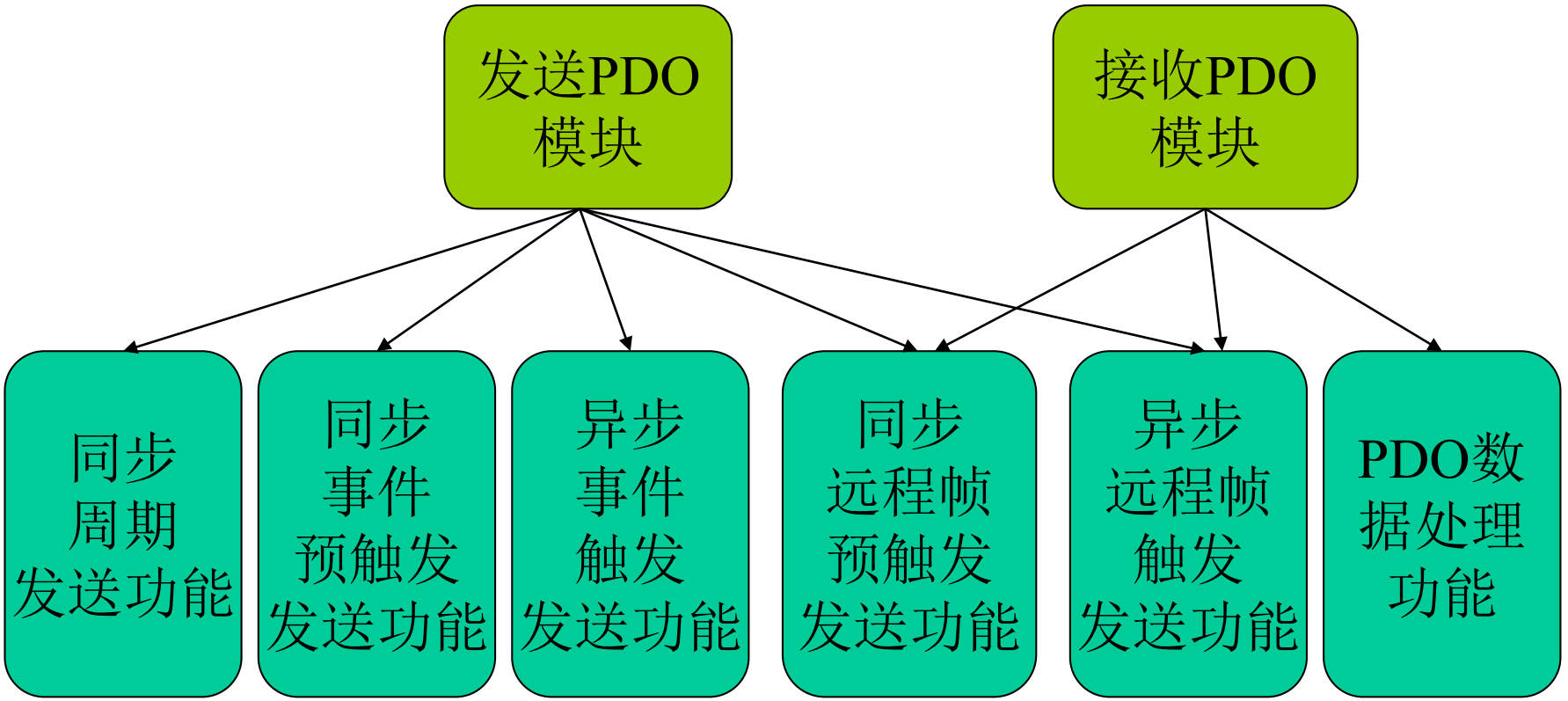
- 实时过程数据传输模块组成



- 建立PDO函数
  - 被发送和接收模块调用。
  - 根据映射参数表建立PDO报文。



- 发送和接收模块
  - 实现PDO报文的各种传输方式。
  - 处理收到的PDO报文。

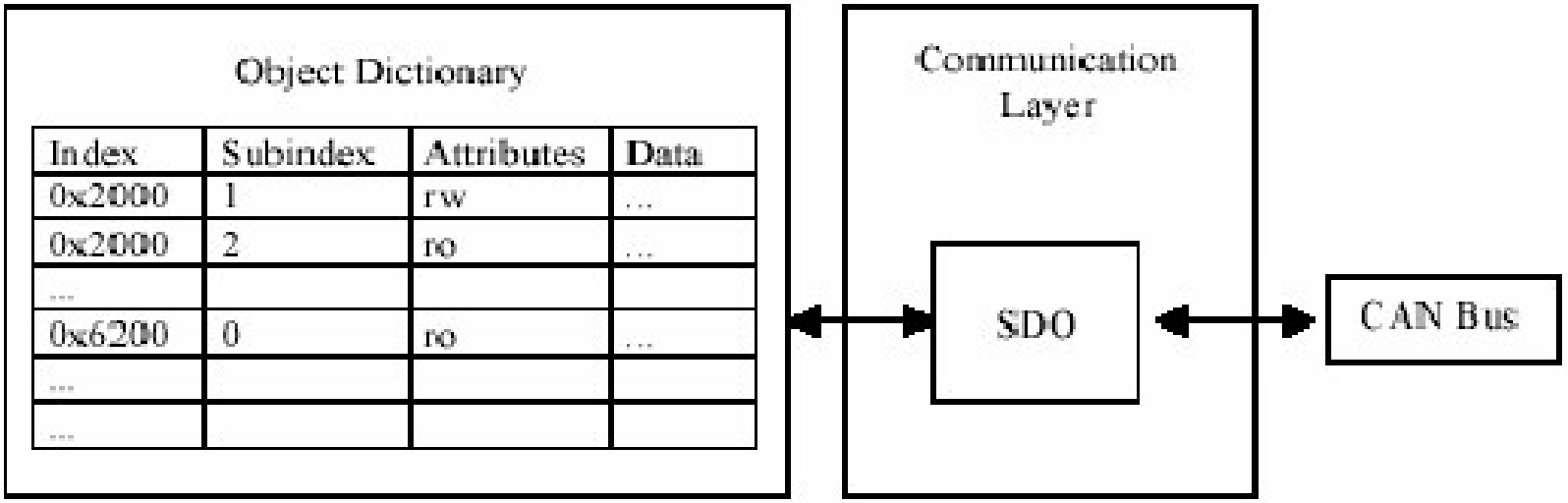


2.3.6 参数读取和设置

管理者如何配置和读取节点服务的参数？

利用CANopen协议的SDO (Service Data Object)功能。

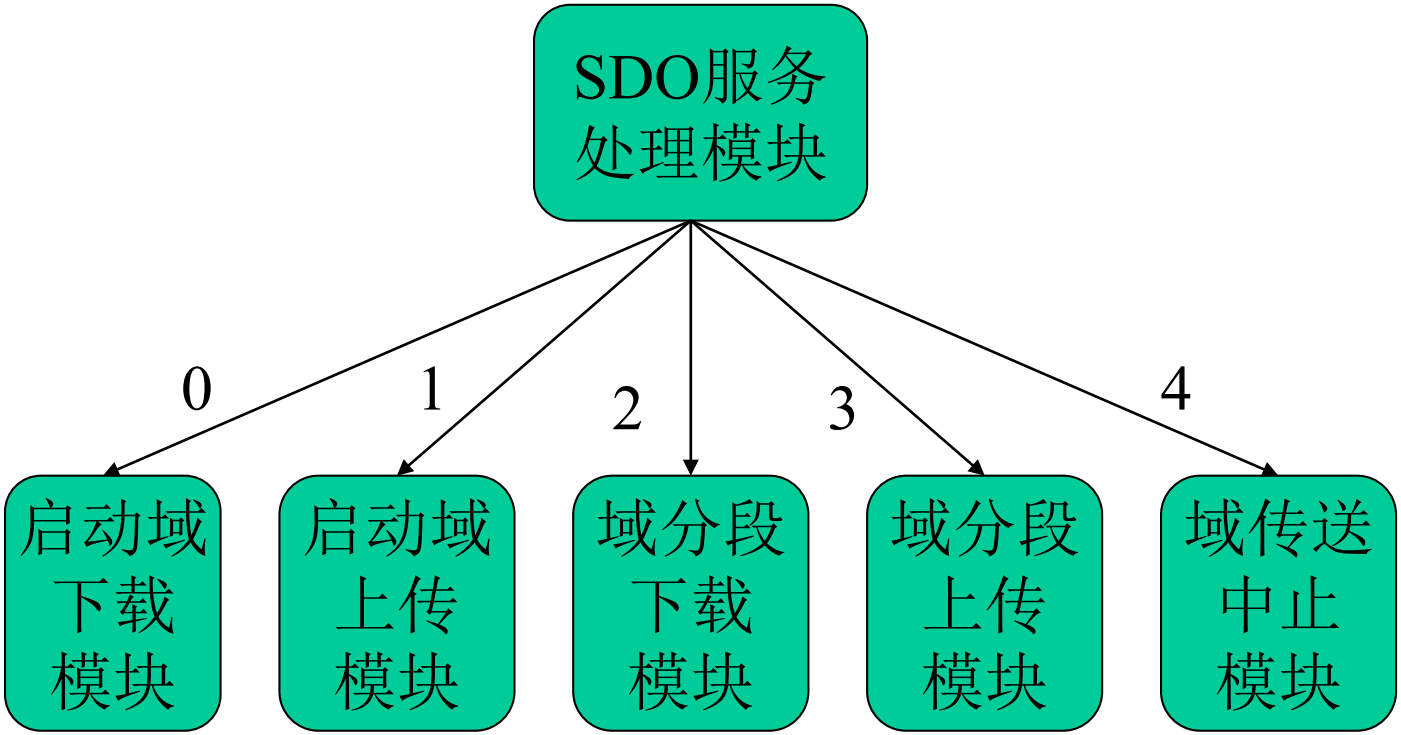
- 对象字典的远程访问接口
- 用来配置和读取节点的参数
- 基于客户端/服务器模式



- 对象字典安全要求高，数据项长度不一，访问须遵循一定的规范。
- SDO服务定义了5个协议：启动域下载，域分段下载，启动域上传，域分段上传和域传送中止。
- 访问协议体现在SDO报文命令字中。

- SDO服务实现

关键是解析SDO命令字。命令字前三位只有0、1、2、3、4这几种类型，对其进行分类并按照协议规范处理，SDO功能即可以实现。



2.3.7 紧急事件服务

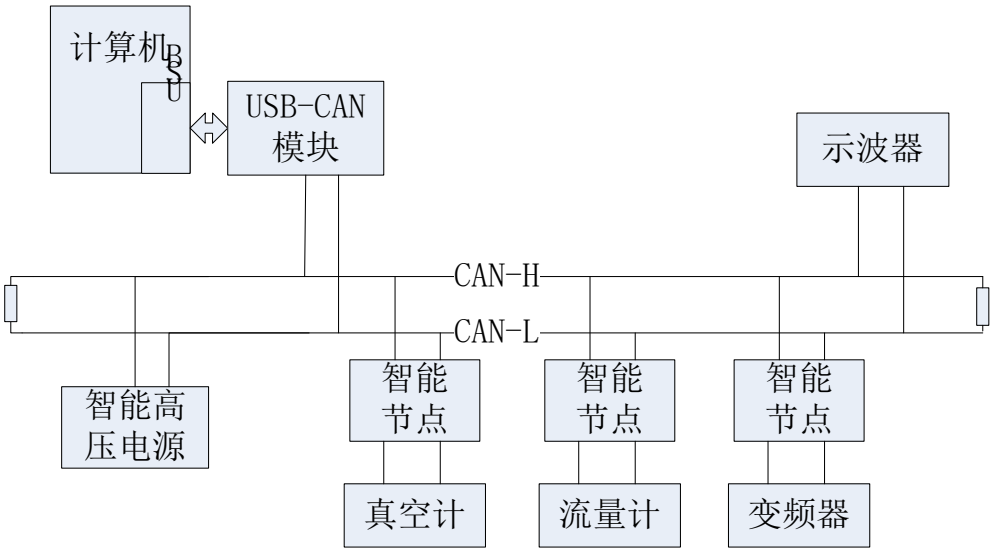
- 属于网络管理的一部分
- 嵌入到用户应用程序中
- 若检测到紧急情况（过压、过流等），处理过程分两步：

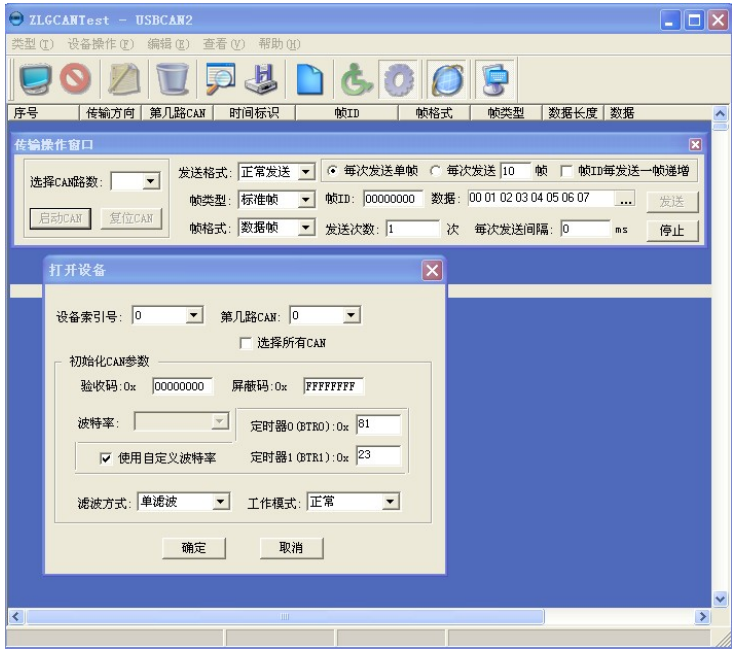
1. 采取既定的紧急保护措施，断电保护、输出复位。
2. 以高优先级发送紧急事件报文，携带节点地址和错误代码，向管理者汇报报警信息，等待后续处理。

2.4 协议栈测试

2.4.1 测试与实验平台搭建

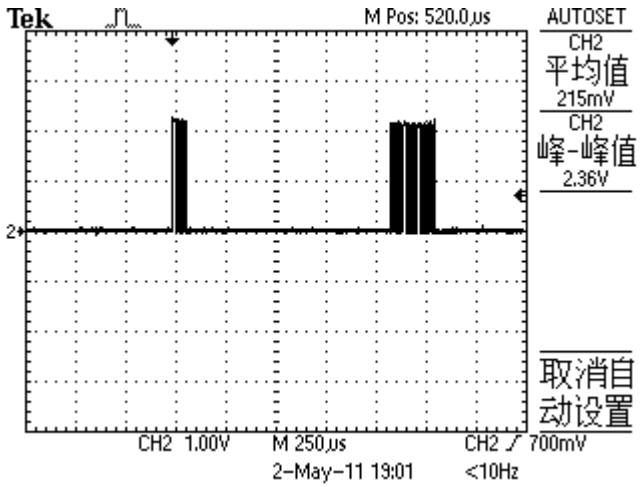
- USB-CAN模块作为监控计算机的CAN网络接口
- 具备存储功能的示波器观察总线波形
- 挂载4个智能节点，囊括了基本的低温等离子体设备
- ZLGCANtest软件监视总线上所有报文
- 总线波特率500Kbps



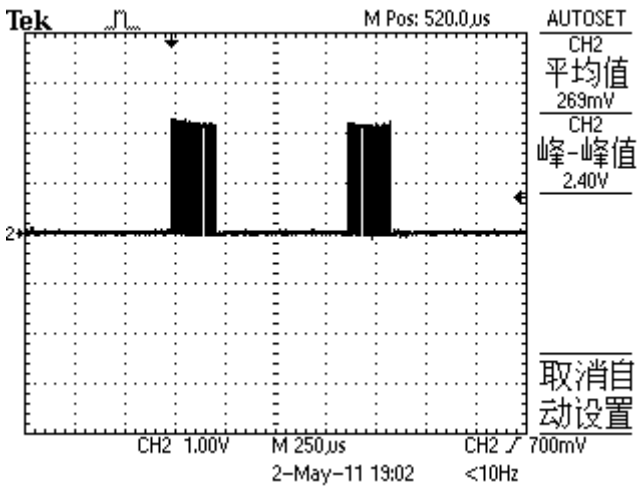


2.4.2 单节点通信周期测试

- 单节点实时过程数据传输最小周期测试
  - 计算机发送同步报文后，节点根据PDO通信参数发送报文。
  - 设置PDO为同步周期发送模式，周期为”1”



- 单节点参数配置通信最小周期测试
  - 计算机发送SDO请求，从节点接收到后必须回复一个SDO应答，测试参数配置响应周期。



2.4.3 多节点实时数据通信周期测试

- 模拟多节点CAN网络实际运行的情况
- 测试最小的可靠网络数据刷新时间
- 观测不同通信周期下的丢包现象

同步周期	同步报文数量	接收PDO数量	延迟发送和丢包现象
100ms	1000	4000	正常无延迟
40ms	1000	4000	正常无延迟
30ms	1000	4000	开始出现延迟发送现象
25ms	1000	4000	延迟发送现象加剧
20ms	1000	4000	延迟发送现象更严重
15ms	1000	1973	出现严重丢包现象
10ms	1000	2052	出现严重丢包现象

2.4.4 实现功能总结

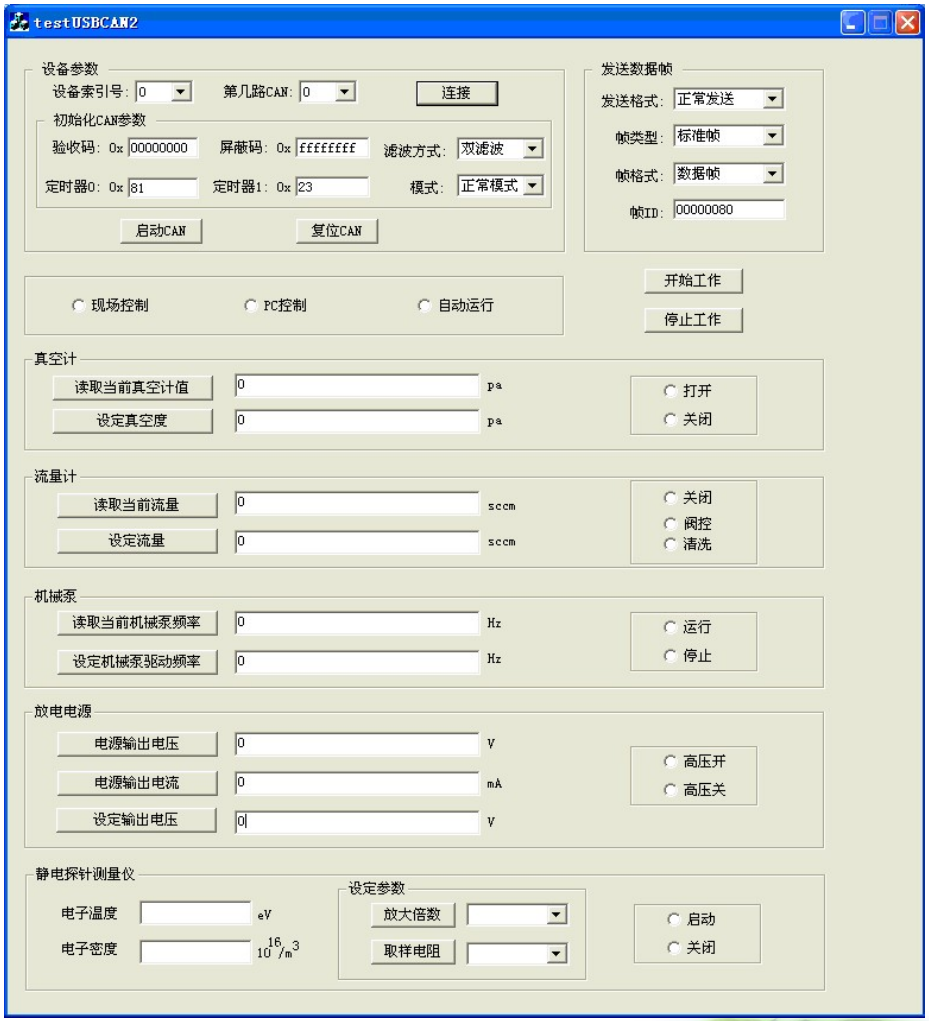
- 实现的协议栈提供的功能如下表所示：

功能	必要性	本文协议栈
SDO服务器端	必要	完成，支持1个服务器端
SDO客户端	必要	部分支持
PDO发送	必要	完成
PDO接收	必要	完成
节点保护/心跳保护	必要	支持心跳保护
运行状态机	必要	完成
紧急事件报文	必要	完成，通知上层
时间戳	可选	不支持
自动设置CAN波特率	可选	不支持
通过拨码开关设置节点号	可选	不支持
可移植性		C编写，模块化，可移植性较好

2.4.5 应用

将CANopen与各设备软件功能模块结合，实现了设备的计算机控制、数据实时汇报等功能。

相比原来的手动控制有效提高了设备控制精度和监控效率，对于提高低温等离子体设备系统的自动化水平有积极的借鉴意义。



2.4.6 低温等离子体设备系统中主从节点帧格式定义

- 从站帧格式：

真空计									
报文种类	cob-id	1	2	3	4	5	6	7	8
控制报文	202H	手/自动	启/停						
汇报报文	182H	真空度（低8位）	真空度（高8位）	手/自动状态	启/停状态				
流量计									
控制报文	203H	手自动切换	关闭/阀控/清洗	流量（低8位）	流量（高8位）				
汇报报文	183H	手/自动状态	关闭/阀控/清洗状态	流量（低8位）	流量（高8位）				

变频器									
控制报文	204H	运行/停止	手/自动切换	驱动频率（低8位）	驱动频率（高8位）				
汇报报文	184H	运行/停止状态	手/自动状态	驱动频率（低8位）	驱动频率（高8位）	驱动电压（低8位）	驱动电压（高8位）	驱动电流（低8位）	驱动电流（低8位）
高压电源									
控制报文	201H	启动/停止供电	手/自动切换	自动控制输出（00H-64H）					
汇报报文	181H	输出电流（低8位）	输出电流（高8位）	输出电压（低8位）	输出电压（高8位）				

- 主站对象字典中的PDO映射参数(主站的，注意跟从站的区分)

1.1、TPDO（发送报文，传输控制命令）

1.1.1 TPDO1（与高压电源通信） COB_ID :201H

6200 01 08 启/停控制 （00启动，01断开）

6200 02 08 手自动切换（00 单片机控制 ，01 手动控制）

6410 01 08 自动控制输出（00H-64H，对应数字电位器5-0V）

1.1.2 TPDO2 (与真空计通信) COB_ID :202H

6200 03 08 手/自动控制

6200 04 08 启停控制

1.1.3 TPDO3 (与流量计通信) COB_ID :203H

6200 05 08 手/自动切换

6200 06 08 关闭/阀控/清洗

6411 01 10 流量设定值（16位）

1.1.4 TPDO4 (与变频器通信) COB_ID :204H

6200 07 08 运行/停止

6200 08 08 手自动切换

6411 02 10 驱动频率（16位）

1.2、RPDO（汇报报文，接收I/O数据）

1.2.1 RPDO1（与高压电源通信） COB_ID :181H

6401 01 10 电流反馈（16位）

6401 02 10 电压反馈（16位）

1.2.2 RPDO2 (与真空计通信)　COB_ID :182H

6401 03 10 真空度反馈（16位）

6000 01 08 手/自动状态

6000 02 08 启/停状态

1.2.3 RPDO3 (与流量计通信) COB_ID :183H

6000 03 08 手/自动状态

6000 04 08 关闭/阀控/清洗状态

6401 04 10 流量设定值（16位）

1.2.4 RPDO4 (与变频器通信) COB_ID :184H

6000 05 08 运行/停止状态

6000 06 08 手/自动状态