消息与信号

消息，数据容器

数据标识　　传输的数据块（最多8字节）　　用符号描述

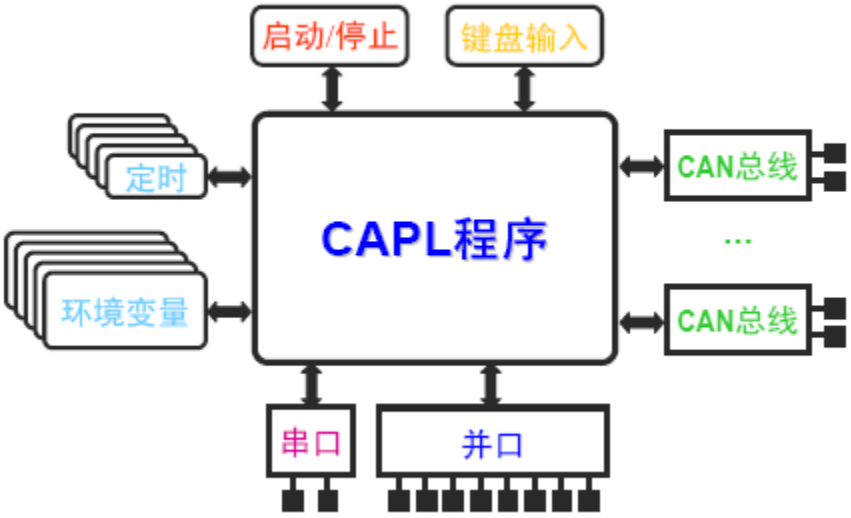
信号，实际使用的信息

信号长度可能从1位到多字节

需要物理单位，需要转换单位　　对错误的描述　　用符号描述

CAPL编程

- CAPL是CAN总线访问编程语言（CAN Access Programming Language ）
- 类C语言
- 应用于Vector CAN工具节点编程
- 基于事件建模的语言
  - 总线事件
  - 属性事件
  - 时间事件



应用

（1）节点仿真

（2）网络仿真

（3）仿真控制系统的环境

（4）节点测试

（5）网关

CAPL程序对事件的响应

CAPL程序能够检测事件，并执行和事件相关的程序。检测的事件类型包括：程序开始执行事件程序停止执行事件□键盘输入事件□CAN消息的接收事件□定时器超时事件□图形面板输入事件（该项只在CANoe中应用）□CAPL程序是基于事件程序的组合

CAPL 事件的基本类型

事件类型	事件名	程序执行条件	事件过程语法结构 *
系统事件	*PreStart*	CANoe初始化时执行	*on preStart { … }*
	Start	测量开始时执行	*on start { … }*
	StopMeasuremet	测量结束时执行	*on stopMeasurement { … }*
CAN控制器事件	*BusOff*	硬件检测到BusOff时执行	*on busOff { … }*
	ErrorActive	硬件检测到ErrorActive时执行	*on errorActive { … }*
	ErrorPassive	硬件检测到ErrorPassive时执行	*on errorPassive { … }*
	WarningLimit	硬件检测到WarningLimit时执行	*on warningLimit { … }*
CAN消息事件	*自定义*	接收到指定的消息时执行	*on message Message { … }*
时间事件	*自定义*	定时时间朝过时执行	*on timer Timer { … }*
键盘事件	*自定义键值*	指定的键被下时执行	*on key Key { … }*
错误帧事件	*ErrorFrame*	硬件每次检测到错误帧时执行	*on errorFrame { … }*
环境变量事件	*自定义*	指定的环境变量值改变时执行	*on envVar EnvVar { … }*

* “事件过程语法结构”列中兰色字体表示该程序的关键字；深红色字体表示用户自定义的名称；“{…}”内是CAPL程序体，用户可根据需要使用CAPL语言编写。

消息过程

```
on message123 //对消息123(dec)反应
on message0x123 //对消息123(hex)反应
on messageMotorData//对消息MotorData(符号//名字)反应
```

```
on messageCAN1.123 /*对CAN 通道1收到消息123反应*/
on message* //对所有消息反应
on message100-200 //对100-200间消息反应
```

键盘过程

```
on key'a'//按'a'键反应
on key''//按空格键反应
on key0x20 //按空格键反应
on keyF1 //按F1键反应
on keyCtrl-F12 //按Ctrl + F12键反应
on keyPageUP//按PageUp键反应
on keyHome //按Home键反应
on key* //按所有键反应
```

时间过程

- 时间过程表示法：
- on timermyTimer//对myTimer设定的时间到反应
- 定时器的申明
- msTimermyTimer;//将myTimer申明ms为单位的变量
- timermyTimer;//将myTimer申明s为单位的变量
- 定时器的设置
- setTimer(myTimer,20);//将定时值设定为20ms，并启动
- cancelTimer(myTimer);//停止定时器myTimer

每次使用etTimer的设置，只能触发一次时间过程

环境变量过程

环境变量过程on envVar对环境变量值的改变产生反应

测量设置中的CAPL节点不会阻止环境变量在数据流图中的传输

环境变量过程常用的函数：

```
getValue()//获取环境变量的值
putValue()//设置环境变量的值
```

可使用this在过程内部访问环境变量的值

数据类型

无符号整数

byte（1字节） word（2字节） dword（4字节）有符号整数 int（2字节）

long（4字节） 浮点数 float（8字节） double（8字节）

CAN消息类型

message 定时器类型 timer（秒为单位） msTimer（毫秒为单位） 单个字符 char（1字节）

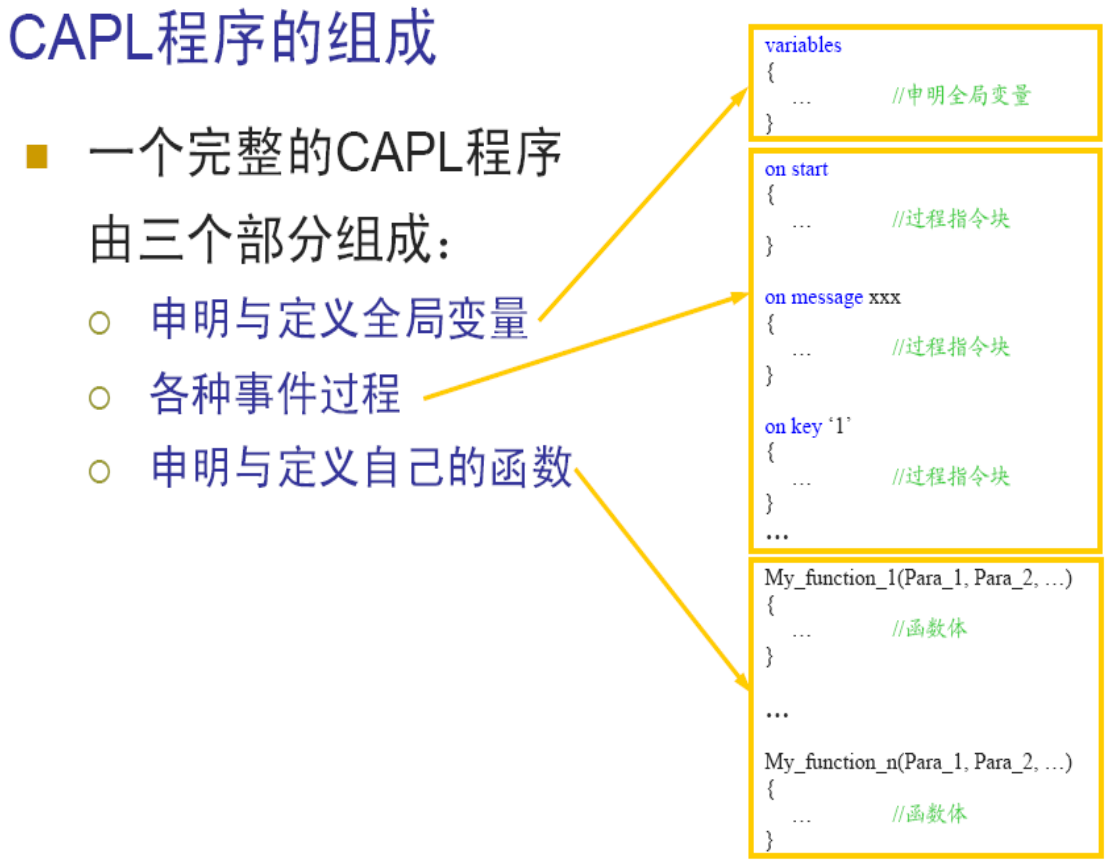
消息的声明

消息申明的格式

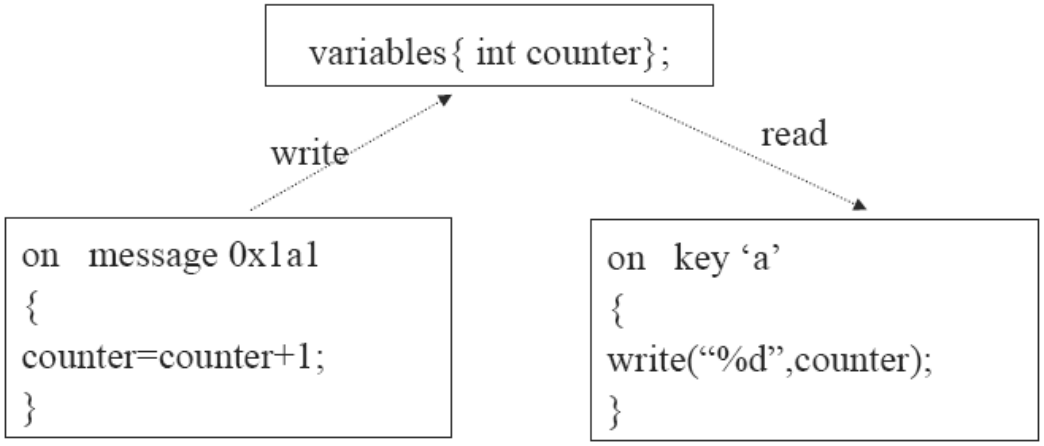
```
Message 0xA my_msg1;
Message 100 my_msg2;
MessageEngineData my_msg3;
```

消息数据的索引

```
my_msg1.byte(0)//数据字节0
my_msg2.word(2)//从第2字节开始的一个字
my_msg3.EngSpeed/*如果使用了符号数据库，可使用信号符号名来索引消息中的信号*/
```



程序执行顺序



事件过程之间无关联，执行顺序由运行时间事件决定

事件过程通过全局变量和子程序决定

事件过程为一整体，不能被其它事件中断

针对消息的一些常用语句

常进行读写

```
if (this.id==100)
{...}//消息ID
```

常写的

```
msg.can=2;//消息所使用的CAN控制器编号
msg.dlc=8;//消息中包含的数据字节长度
```

常读的

```
dwordt ; t=this.time;//消息的时标，单位是10us
if(this.dir!=RX) {return;}//消息
的收发特性
```

注意： **this this** 是关键字，在事件过程中代表所定义的触发事件名

关键字 —— this

- 在事件过程中，关键字this指定事件对象的数据结构

```
on message 100 {
        byte byte_0;
        byte_0 = this.byte(0);
        ...
}
```

```
on envVar Switch {
        int val;
        val = getvalue(this);
        ...
}
```

- this可作为参数使用
- 对于this值的改变仅在过程内部有效

CAPL 指令块

```
Counter=counter+1;
If(counter==256)
{
Counter=0;
Stop();
}
```

CAPL 中输出文本

```
Int h=100;
Char ch='a';
Char s100[8]="hundred";
Write("Hundred as a number:%d,%x",h,h);
Write("Hundred as a string:%s,%x",s100);
Write("Hundred as a number:%6.4g",sqrt(2.0))
```

处理信号：

```
On message 0x64
{
   If(this.byte(2)=0xFF);
```

```
    Write(“third byte of message is invalid”);
}
On message MotorData
{
   If(this.temperature.phys>=150);
    Write(“warning:critical temperature”);
}
```

传输信号

```
on key ‘a’ {
    message MotorData mMoDa;
    mMoDa.temperature.phys=60;
    mMoDa.speed.phys=4300;
    output(mMoDa);
}
on key ‘b’ {
     message 100 m100= {dlc=1};
     m100.byte(0)=0x0B;
     output(m100);
}
```

周期性消息发送的CAPL示例

```
Variables                              // 定义全局变量
{
      message 0x555 msg1 = {dlc=1}; // 定义消息变量 msg1，并初始化数据字节代码为1
      msTimer timer1;                  // 定义定时器变量 timer1
}
on start                               // 系统过程
{
      setTimer(timer1,100);            // 初始化定时器变量timer1的值为 100 msec，并启动
}
on timer timer1                        // 时间过程（对于定时器变量timer1 ）
{
      setTimer(timer1,100);            // 重新设置timer1 ，并启动
      msg1.byte(0)=msg1.byte(0)+1; // 改变消息数据字节
      output(msg1);                    // 输出消息
}
```

环境变量过程的示例

```
Variables
{
}

// Reaction to change of environment var. evSwitch
on envVar evSwitch
{
      // Declare a CAN message to be transmitted
      message Msg1 msg;
      // Read out the value of the light switch, Assign to the bus signal bsSwitch
      msg.bsSwitch = getValue(this);
      // Output message on bus (spontaneous transmission)
      output(msg);
}
```