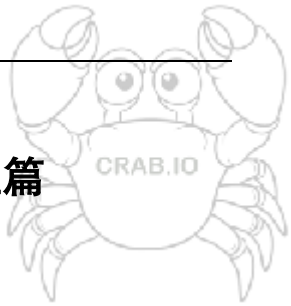


大脚蟹快速开发平台学习教程《三》：高级编程篇

上一节，我们学习了《基础入门篇》，本篇将学习进阶语法，和高级开发方式。

本文默认所有的工具已经准备就绪中。

本文建议大家使用 **SublimeText 3.0**，不仅界面清爽美观，而且功能强大好用。

一、事件

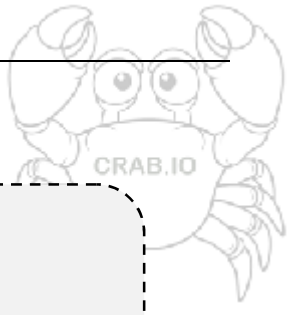
事件是指当系统因某个条件达成，或是硬件触发而发生的任务函数。事件的触发和执行是按顺序的。一旦开发者定义了事件的触发函数，则系统将中断当前的任务，跳转到事件函数，等待事件执行完毕，才返回到当前被中断的任务中。

例外的情况是，如果当前的任务是在事件函数中，则任务不会被中断，必须等待当前事件函数执行完毕，才会跳转到下一个事件函数。如果当前的任务是在执行系统的硬件驱动程序，那也同样不会被中断，直到硬件驱动程序返回用户程序中，才会发生事件任务跳转。

1）事件函数的定义，语法如下：

```
event <Event Name>: <Event ID>
{
   Method Body
}
```

- `Event Name`：事件函数名称，这是一个可选项。如果需要这个名称，则这个名称必须是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- `Event ID`：事件标志 **ID**，这个 **ID** 是指示当系统发生某个触发条件时所对应的标志号。
- **Method body**：函数主体，包含了完成任务所需的指令集。
- 事件函数可以使用 **return** 语句，但不可以返回任何值。同样，**Result** 不起作用，所以也是无效的变量。
- 在事件函数里， 不得使用 **OpenEvent()**和 **CloseEvent()**函数。
- 如果在事件函数有死循环语句，那么它将永远跳出不该事件。
- 开发者应该避免在事件函数里处理那些比较消耗时间和消耗资源的任务语句。
- 如果在即将发生的事件函数中修改了某个全局函数，而被中断的当前任务中，也使用了同样的全局函数，那么，在事件函数发生后，返回到当前任务时，将会发生不可预测的问题。
- 在同一个应用程序里，事件标志 **ID** 不可以重复，否则事件任务的跳转将不可预测。
- 除了以上的事件，其它的使用方式和普通函数没有区别。



范例：

```
//当用户按下F1按键的时候，将会进入下面的事件函数
//事件函数：OnKey1              这个函数名是可选的。
//事件标志：KEY_PRESS_F1     每一种事件都有自己惟一的标志
event  OnKey1 :  KEY_PRESS_F1
{
  board.LED2 = LED_RED;
  Delay(500);     //这里延时500毫秒，仅仅是为了看清LED是否已经点亮
  board.LED2 = LED_OFF;
}

//当用户使用遥控器的时候，将会进入下面的事件函数
//事件函数：没有函数名，系统将会自动按标志起名
//事件标志：REMOTE_CONTROL
event  REMOTE_CONTROL
{
  ushort  RemoteKey;
  RemoteKey = Board.RemoteKey;
  PrintLn("Remote Control is: " # RemoteKey);
}
```

二、接口

接口是应用程序与硬件驱动程序交互与沟通的主要通道。任何需要控制硬件的程序，都需要经过接口的定义来操作和读写。接口定义的语法如下：

```
interface  <Interface Parent>::<Interface Name>
{
  default extern
  {
    get:  <API default get function name>;
    set:  <API default set function name>;
  }

  property <Data Type> <Property name>
  {
    get :  <API get function name>;
    set :  <API set function name>;
    index:  <API port index>;
    type:  <API port type>;
  }

  extern <Return Type> <Method Name>(Parameter List) :  < API Name>;
}
```

1）接口的定义

- `interface`：表示当前的定义是接口类型。接口是一个集合类型，它可以包函多个接口属性和多个接口函数。
- `Interface Parent::`当前所要定义的接口所属的父类接口。父类接口必须是已经定义过的，已经存在的接口。该接口是可选项，如果忽略不写，则父类接口是默认的系统全局接口。
- `Interface Name:` 接口名称，它是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。

2)接口属性缺省值定义

- **default extern**：表示当定所定义的是接口属性缺省值。
- **get**：接口属性缺省的读方法定义。
- `API default get function name`：接口属性缺省的读方法的 **API** 函数名称，必须是字符串。该项是可选项，如果忽略，则缺省的函数名称是 **get+**接口名称。
- **set**：接口属性缺省的写方法定义。
- `API default set function name`：接口属性缺省的写方法的 **API** 函数名称，必须是字符串。该项是可选项，如果忽略，则缺省的函数名称是 **set+**接口名称。

3）接口属性的定义

- property：表示当前所定义的是接口属性。
- Data Type：当前接口属性的数据类型，它可以是任何值类型，包括字符串。但不可以是数组，接口和类。
- Property Name：接口属性名称，它是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- **get**：接口属性的读方式定义，表示当前的接口属性提供读的方法。
- API get function name：接口属性的读方法的 **API** 函数名称，必须是字符串。该项是可选项，如果忽略，但又定义了 **get**，则 **API** 函数名称将会采用接口属性缺省值所定义的 **get** 方式的名称，如果没有定义接口属性缺省值，则缺省的函数名称是 **get+**属性名称。
- **set**：接口属性的写方式定义，表示当前的接口属性提供写的方法。
- API set function name：接口属性的写方法的 **API** 函数名称，必须是字符串。该项是可选项，如果忽略，但又定义了 **set**，则 **API** 函数名称将会采用接口属性缺省值所定义的 **set** 方式的名称，如果没有定义接口属性缺省值，则缺省的函数名称是 **set+**属性名称。
- index：接口属性的端口索引号定义，与 API port index 配套使用。
- API port index：接口属性的端口索引号，必须是无符号整型。该项是可选项，如果忽略，则默认是 0。
- type：当前接口属性的类型定义。此定义将会改变接口属性的读写指令，仅系统使用。
- API port type：当前接口属性的类型，必须是字符串。该项是可选项，默认是“PORT”。

4）接口函数的定义

- extern：定义本函数是接口类型的函数，接口函数只有函数名称和参数定义，没有函数本身。
- Return Type：返回类型，一个函数可以返回一个值。返回类型是函数返回的值的数据类型。如果函数不返回任何值，则返回类型为 **void**。
- Method Name：函数名称，是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- Parameter List：参数列表，使用圆括号括起来，该参数是用来传递和接收函数的数据。参数列表是指函数的参数类型、顺序和数量。参数是可选的，也就是说，一个函数可能不包含参数。
- API name：接口函数所对应的硬件驱动 API 函数名称。该项是可选项，如果忽略，则该函数所对应的 API 函数名称就是函数本身的名称。
- 接口函数所应的硬件驱动 API 函数必须是存在的，否则在调用的时候，将引起系统异常发生，从而使应用程序停止运行。

4）接口的定义过程

接口一般由硬件驱动的开发者，或是模块厂商提供，应用程序开发者不应自己修改接口定义代码。接口本身不占用内存，但接口属性和接口函数所对应的 **API** 函数，会占用 **API** 函数接口列表（请参阅《系统移植篇》）。

范例：

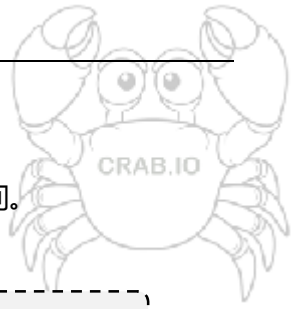
```
//定义一个接口，名称为Board
interface Board
{
  //定义一个Key属性，仅支持读方法，其对应的读API 函数为'getKey'
  property ushort Key  { get : 'getKey'; }

  //定义一个LED1属性，仅支持写方法，其对应的读API 函数为'setLED'，
  //并且所对应的端口索引为1
  property byte LED1  { set : 'setLED'; index: 1; }

  //定义一个LED2属性，仅支持写方法，其对应的读API 函数为'setLED'，
  //与上面的LED1共享同一个API 函数，并且所对应的端口索引为2
  property byte LED2  { set : 'setLED'; index: 2; }

  //定义一个Beep接口函数，接受两个参数。
  //硬件驱动API 函数名称为'Beep'
  extern void Beep(byte Width, byte Count);
}
```

5）使用接口

接口一经定义，无须经过变量声明的方式，便可直接使用。

范例：

```
ushort K1;
K1 = board.Key;  //读取开发板按键码
Print(K1);  //显示按键码
board.LED1 = LED_RED;  //设置开发板LED1为红色

//让开发板蜂鸣器工作，频宽为1400，次数为3，也就是响三次。
board.Beep(1400, 3);
```



三、枚举

枚举类型是一系列的常量定义的集合，它相当於给一些常量定义提供一个名称空间。

1）枚举类型的定义方式，语法如下：

```
enum <EnumGroup Name>
{
   <Enum Name> = <Value>
}
```

- enum：表示当前定义的枚举类型。
- EnumGroup Name：枚举类型集合的名称，它是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- Enum Name：当前枚举类型集合里的单项枚举类型名称，它是一个唯一的标识符，大小写不敏感的。它不能与当前集合里声明的其他标识符相同。
- **Value**：一个常数值，它必须是一个值类型（包括字符串），或者是常量表达式。该值是可选项，如果忽略该值，则当前的枚举值将是上一个枚举值+1。如果上一个枚举值是非数字型，则当前枚举值无效。如果当前的枚举项是第一项，且又如果忽略该值，则默认该值为 0。

范例：

```
enum Weekdays
{
  Sunday = 0,
  Monday = 1,
  Tuesday = 2,
  Wednesday = 3,
  Thursday = 4,
  Friday = 5,
  Saturday = 6
}
```

2）枚举类型的使用

枚举类型一经定义，无须经过变量声明的方式，便可直接使用。使用时，必须联合枚举集合名和枚举项一起使用。

```
int V1 = Weekdays.Sunday;

V1 = Weekdays.Monday + Weekdays. Tuesday
```

四、辅助接口

辅助接口是一个特殊的接口，它的作用是给变量提供辅助函数，以便获得变量的其它信息，或是将变量转换成其它类型。接口定义的语法如下：

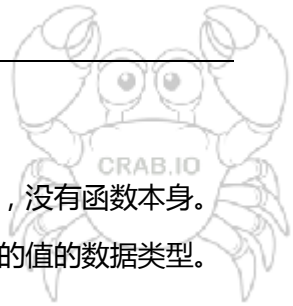
```
helper  <Helper Name>
{
  property <Data Type> <Property name>
  {
    get : <API get function name>;
    set : <API set function name>;
  }

  extern <Return Type> <Method Name>(Parameter List) : < API Name>;
}
```

1）辅助接口的定义

- helper：表示当前的定义是辅助接口类型。接口是一个集合类型，它可以包函多个接口属性和多个接口函数。
- Helper Name: 辅助接口名称，它必须与值类型定义的名称相同，大小写不敏感的。相同的辅助接口名称将会归类到同一个值类型定义的辅助接口里。

2）辅助接口属性的定义

- property：表示当前所定义的是接口属性。
- Data Type：当前接口属性的数据类型，它可以是任何值类型，包括字符串。但不可以是数组，接口和类。
- Property Name：接口属性名称，它是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- **get**：接口属性的读方式定义，表示当前的接口属性提供读的方法。
- API get function name：接口属性的读方法的 **API** 函数名称，必须是字符串。该项是可选项，如果忽略，但又定义了 **get**，则 **API** 函数名称将会采用接口属性缺省值所定义的 **get** 方式的名称，如果没有定义接口属性缺省值，则缺省的函数名称是 **get+**属性名称。
- **set**：接口属性的写方式定义，表示当前的接口属性提供写的方法。
- API set function name：接口属性的写方法的 **API** 函数名称，必须是字符串。该项是可选项，如果忽略，但又定义了 **set**，则 **API** 函数名称将会采用接口属性缺省值所定义的 **set** 方式的名称，如果没有定义接口属性缺省值，则缺省的函数名称是 **set+**属性名称。
- 辅助接口属性不允许有 **index** 和 **type** 的定义。



3）辅助接口函数的定义

- `extern`：定义本函数是接口类型的函数，接口函数只有函数名称和参数定义，没有函数本身。
- `Return Type`：返回类型，一个函数可以返回一个值。返回类型是函数返回的值的数据类型。如果函数不返回任何值，则返回类型为 **void**。
- `Method Name`：函数名称，是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- `Parameter List`：参数列表，使用圆括号括起来，该参数是用来传递和接收函数的数据。参数列表是指函数的参数类型、顺序和数量。参数是可选的，也就是说，一个函数可能不包含参数。
- `API name`：接口函数所对应的硬件驱动 API 函数名称。该项是可选项，如果忽略，则该函数所对应的 API 函数名称就是函数本身的名称。
- 接口函数所应的硬件驱动 API 函数必须是存在的，否则在调用的时候，将引起系统异常发生，从而使应用程序停止运行。

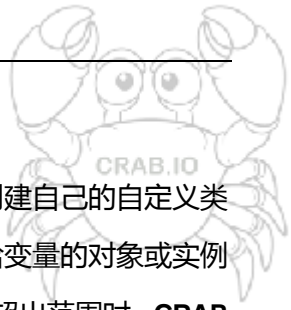
范例：

```
//定义一个int类型的辅助接口
helper int
{
  //定义一个辅助接口属性，仅支持读方法，其对应的读API函数为'getIntAbs'
  property int Abs
  {
    get:'getIntAbs';
  }

  //定义一个接口函数，返回值是字符串，其对应的读API函数为'IntToString'
  extern string ToString() : 'IntToString';
}
```

4）辅助接口的使用

范例：

```
int V1, V2;
V1 = -10;
V2 = V1.Abs();
Print(V2.ToString());
```



五、类

类属于一种构造集合，使用类，可以通过组合其他类型的变量、函数和属性创建自己的自定义类型。 类就象蓝图， 它定义类型的数据和行为，客户端代码就可以通过创建分配给变量的对象或实例来使用该类。 变量会一直保留在内存中，直至对变量的所有引用超出范围为止。 超出范围时，**CRAB**将对其进行标记，以便用于内存垃圾回收。

1）类定义

类相当一个命名空间，在这个命名空间里面，可以有常量定义，变量定义，属性定义，成员函数。

语法如下：

```
class  <Class Name>
{
  const <constant name> = <value>;

  <data_type> <variable_list>;

  property <Data Type> <Property name>
  {
    get : { get function body }
    set : { set function body }
  }

  <Access Specifier> <Return Type> <Method Name>(Parameter List)
  {
     Method Body
  }
}
```

2）常量定义和使用

类空间的常量定义格式，和全局空间的常量定义方式一致，详细定义方法参照《基础入门篇》。

如果需要使用，则视使用地点而变。如果是在类空间里，而直接使用常量名称即可。如果是在全局空间，或是其它类空间里，而需要在前面加上类名称或是对象名称。

3）变量的定义和使用

类空间的变量定义格式，和全局空间的变量定义方式一致，详细定义方法参照《基础入门篇》。

如果需要使用，则视使用地点而变。如果是在类空间里，而直接使用变量名称即可。如果是在全局空间，或是其它类空间里，而需要在前面加上对象名称。类空间的变量，是寄生在对象内存里的，只有为对象申请内存之后，类变量才会真实存在。

4）类属性的定义

- `property`：表示当前所定义的是类属性。
- `Data Type`：当前接口属性的数据类型，它可以是任何值类型，包括字符串。但不可以是数组，接口和类。
- `Property Name`：类属性名称，它是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- **get**：类属性的读方式定义，表示当前的类属性提供读的方法。
- `get function body`：类属性的读函数，它没有任何参数，而且必须返回一个值，就是有类属性相同的数据类型的值。和普通函数一样，它也自带 **Result** 变量，可以当作返回值使用。
- **set**：类属性的写方式定义，表示当前的类属性提供写的方法。
- `set function body`：类属性的写函数，它没有任何参数，而且也不能返回任何值，但它自带一个和类属性相同的数据类型的变量 **Value**。这个变量 **Value** 的值，是当发生类属性写操作的时候，由系统传入的。
- 类属性一般与类变量配合使用，以达到保护内部变量和计算的作用。

5）类成员函数的定义

- **Access Specifier**：访问修饰符，这个决定了这个类成员函数相对其它类的可见性。在当前的版本里，此修饰符暂时不起作用。
- **Return type**：返回类型，一个类成员函数可以返回一个值。返回类型是函数返回的值的数据类型。如果函数不返回任何值，则返回类型为 **void**。
- **Method name**：函数名称，是一个唯一的标识符，大小写不敏感的。它不能与声明的其他标识符相同。
- **Parameter list**：参数列表，使用圆括号括起来，该参数是用来传递和接收函数的数据。参数列表是指函数的参数类型、顺序和数量。参数是可选的，也就是说，一个函数可能不包含参数。
- **Method body**：函数主体，包含了完成任务所需的指令集。