第 3 章

简洁的 Arduino 语言

Arduino 使用 C/C++语言编写程序，它的语法是建立在 C/C++基础上的，其实就是基础的 C 语法。在一般情况下，C 语言要求一个源程序不论由多少个文件组成，都必须有且仅有一个主函数，即 main()函数。C 语言程序执行是从主函数开始的。但在 Arduino 中，主函数 main()已经在内部定义了，开发者只需要完成 setup()函数和 loop()函数，就能够完成 Arduino 程序的编写。其中，setup()函数负责 Arduino 程序的初始化部分，loop()函数负责 Arduino 程序的执行部分。

下面重点介绍 Arduino 程序的架构、数据类型、数据运算、程序结构、函数的使用。

3.1 语言概览

Arduino 语言很简洁，在官网[①]上可以查询到最新版本的语言参考，图 3-1 所示为

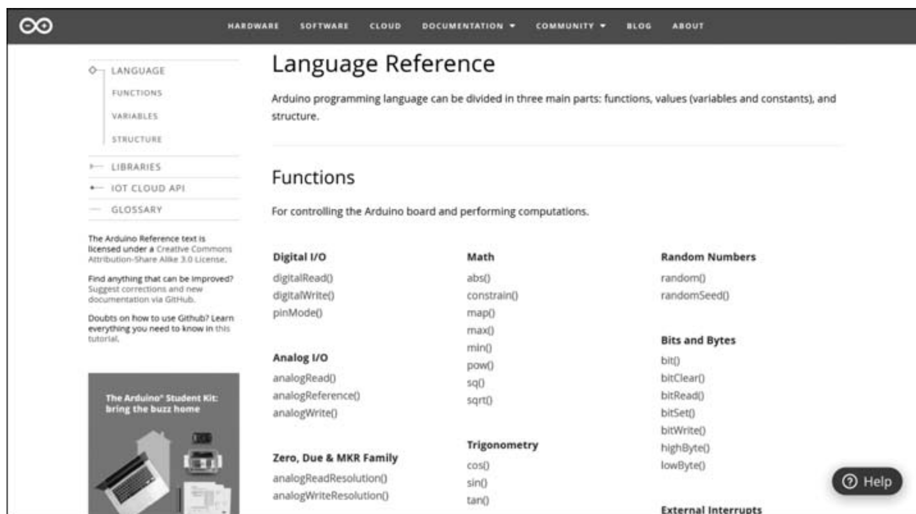

图 3-1　官网的语言参考

① https://www.arduino.cc/reference/en/

Arduino 语言参考 2019 版。

Arduino 语言主要由程序结构、变量、函数三大部分组成，见表 3-1。标注为“Arduino”的为 Arduino 所特有，标注为“标准 C”的与标准 C 语言一致。

表 3-1　Arduino 语言参考

结构、流程、运算		变量、注释		Arduino 函数	
setup()	Arduino	HIGH ｜ LOW	Arduino	数字 I/O	
loop()	Arduino	INPUT ｜ OUTPUT	Arduino	pinMode()	Arduino
程序流程控制		true ｜ false	标准 C	digitalWrite()	Arduino
if	标准 C	整型常量	标准 C	digitalRead()	Arduino
if...else	标准 C	浮点数常量	标准 C	模拟 I/O	
for	标准 C	数据类型	标准 C	analogReference()	Arduino
switch case	标准 C	void	标准 C	analogRead()	Arduino
while	标准 C	boolean	标准 C	analogWrite()	Arduino
do...while	标准 C	char	标准 C	高级 I/O	
break	标准 C	unsigned char	标准 C	shiftOut()	Arduino
continue	标准 C	byte	标准 C	pulseIn()	Arduino
return	标准 C	int	标准 C	计时及延时	
goto	标准 C	unsigned int	标准 C	millis()	Arduino
算术运算符		word	标准 C	delay(ms)	Arduino
＋（加）	标准 C	long	标准 C	delayMicroseconds(μs)	Arduino
－（减）	标准 C	unsigned long	标准 C	数学库	
*（乘）	标准 C	float	标准 C	min()	Arduino
/（除）	标准 C	double	标准 C	max()	Arduino
%（取模）	标准 C	string	标准 C	abs()	Arduino
比较运算符		String(c＋＋)	标准 C	constrain()	Arduino
＝＝（等于）	标准 C	array	标准 C	map()	Arduino
!＝（不等于）	标准 C	数据类型转换	标准 C	pow()	Arduino
<（小于）	标准 C	char()	标准 C	sqrt()	Arduino
>（大于）	标准 C	byte()	标准 C	三角函数	
<=（小于或等于）	标准 C	int()	标准 C	sin(rad)	Arduino
>=（大于或等于）	标准 C	word()	标准 C	cos(rad)	Arduino
布尔运算符		long()	标准 C	tan(rad)	Arduino
&&（逻辑与）	标准 C	float()	标准 C	随机数	
｜｜（逻辑或）	标准 C	变量作用域	标准 C	randomSeed()	Arduino
!（逻辑非）	标准 C	static（静态变量）	标准 C	random()	Arduino
指针运算符		volatile（易变变量）	标准 C	random()	Arduino
*（指针运算符）	标准 C	const（固定变量）	标准 C	位操作	
&（地址运算符）	标准 C	辅助工具	标准 C	lowByte()	Arduino
位运算		sizeof()	标准 C	highByte()	Arduino

续表

<table>
<tr><th colspan="2">结构、流程、运算</th><th colspan="2">变量、注释</th><th colspan="2">Arduino 函数</th></tr>
<tr><td>&（位与）</td><td>标准 C</td><td>扩展语法</td><td>标准 C</td><td>bitRead()</td><td>Arduino</td></tr>
<tr><td>|（位或）</td><td>标准 C</td><td>;（分号）</td><td>标准 C</td><td>bitWrite()</td><td>Arduino</td></tr>
<tr><td>^（位异或）</td><td>标准 C</td><td>{（大括号）</td><td>标准 C</td><td>bitSet()</td><td>Arduino</td></tr>
<tr><td>～（位非）</td><td>标准 C</td><td>//（单行注释）</td><td>标准 C</td><td>bitClear()</td><td>Arduino</td></tr>
<tr><td><<（左移）</td><td>标准 C</td><td>/ ** /（多行注释）</td><td>标准 C</td><td>bit()</td><td>Arduino</td></tr>
<tr><td>>>（右移）</td><td>标准 C</td><td>#define（宏定义）</td><td>标准 C</td><td>设置中断函数</td><td></td></tr>
<tr><td>复合运算符</td><td>标准 C</td><td>#include（包含）</td><td>标准 C</td><td>attachInterrupt()</td><td>Arduino</td></tr>
<tr><td>++（自加）</td><td>标准 C</td><td></td><td>标准 C</td><td>detachInterrupt()</td><td>Arduino</td></tr>
<tr><td>--（自减）</td><td>标准 C</td><td></td><td></td><td>interrupts()</td><td>Arduino</td></tr>
<tr><td>+=（复合加）</td><td>标准 C</td><td></td><td></td><td>noInterrupts()</td><td>Arduino</td></tr>
<tr><td>-=（复合减）</td><td>标准 C</td><td></td><td></td><td>主要串口通信函数</td><td></td></tr>
<tr><td>*=（复合乘）</td><td>标准 C</td><td></td><td></td><td>begin()</td><td>Arduino</td></tr>
<tr><td>/=（复合除）</td><td>标准 C</td><td></td><td></td><td>available()</td><td>Arduino</td></tr>
<tr><td>&=（复合与）</td><td>标准 C</td><td></td><td></td><td>read()</td><td>Arduino</td></tr>
<tr><td>|=（复合或）</td><td>标准 C</td><td></td><td></td><td>flush</td><td>Arduino</td></tr>
<tr><td></td><td></td><td></td><td></td><td>print()</td><td>Arduino</td></tr>
<tr><td></td><td></td><td></td><td></td><td>println()</td><td>Arduino</td></tr>
<tr><td></td><td></td><td></td><td></td><td>write()</td><td>Arduino</td></tr>
<tr><td></td><td></td><td></td><td></td><td>peak()</td><td>Arduino</td></tr>
<tr><td></td><td></td><td></td><td></td><td>serialEvent()</td><td>Arduino</td></tr>
</table>

3.2 Arduino 语言基础

3.2.1 程序的架构

Arduino 程序的架构大体可以分为 3 部分，如程序 3-1 所示。

```
1   /**********************************************************
2    * 程序 3-1: Arduino 程序的架构
3    **********************************************************/
4   int tmpPin = 8; //在最前面定义变量,把引脚号赋值给某变量
5   void setup()
6   {
7       //在这里填写 setup()函数代码,它只运行一次
8   }
9   void loop()
10  {
11      //在这里填写 loop()函数代码,它会不断重复运行
12  }
```

（1）声明变量和接口名称。

（2）setup()。Arduino 程序运行时，首先要调用 setup()函数，一般放在程序开头，用于初始化变量、设置引脚的输出/输入类型、配置串口、引入类库文件等。

每次 Arduino 上电或重启后，setup()函数只运行一次。

（3）loop()。loop()函数用于执行程序，是一个死循环，其中的代码将被循环执行，用于完成程序的功能，如读入引脚状态、设置引脚状态等。

3.2.2　数据类型

Arduino 与 C 语言类似，所有的数据都必须指定数据类型。数据类型在数据结构中的定义是值的集合及在这个值的集合上的一组操作。各种数据类型都需要在特定的地方使用。一般来说，变量的数据类型决定如何将代表这些值的位存储到计算机的内存中，在声明变量时，需要指定它的数据类型，以便存储不同类型的数据。

常用的数据类型有整型、浮点型、布尔型、字符型、字节型、数组及字符串等。

（1）整型。整型即整数类型。Arduino 可以使用的整数类型及取值范围见表 3-2。

表 3-2　Arduino 支持的整数类型及取值范围

整数类型	比特数	取值范围	示例
有符号基本整型 [signed] int	16	−32 768～+32 767	int a=−3；
无符号基本整型 unsigned int	16	0～65 535	unsigned int b = 3267；
有符号长整型 long [int]	32	−2 147 483 648～+2 147 483 647	long c=−4235；
无符号长整型 unsigned long [int]	32	0～+4 294 967 296	unsigned long d=1000；

（2）浮点型。浮点型其实就是平常所说的实数。Arduino 有 float（单精度）和 double（双精度）两种浮点型。浮点数可以用来表示含有小数点的数，如 1.24。float 浮点型数据占 4 字节的内存；double 浮点型数据占 8 字节的内存。双精度浮点型数据比单精度浮点型数据的精度更高。

（3）布尔型。布尔型（boolean）变量的值有两个，即假（false）和真（true）。布尔值是一种逻辑值，可以用来进行计算。最常用的布尔运算符为：与运算（&&）、或运算（||）及非运算（!）。表 3-3 为布尔运算真值表。

表 3-3　布尔运算真值表

A	B	A 与 B	A 或 B	A 非
True	True	True	True	False
True	False	False	True	False

续表

A	B	A 与 B	A 或 B	A 非
False	True	False	True	True
False	False	False	False	True

对于 A&&B,仅当 A 和 B 均为真时,运算结果才为真；否则,运算结果为假。对于 A||B 运算,仅当 A 和 B 均为假时,运算结果才为假；否则,运算结果为真。对于!A 运算,当 A 为真时,运算结果为假；当 A 为假时,运算结果为真。

(4) 字符型。字符型(char)变量可以用来存放字符,数值范围为－128～＋128,如：

```
char A = 58;
```

(5) 字节型。字节型(byte)变量可用 1 字节来存储 8 位无符号数,数值范围为 0～255,如：

```
byte B = 8;
```

(6) 数组。数组是由一组具有相同数据类型的数据构成的集合。数组中的每一个数据都具有相同的数据类型,可用一个统一数组名和下标来唯一确定数组中的每个数据。Arduino 的数组是基于 C 语言的。本节只简单介绍如何定义和使用数组。

数组的声明和创建与变量一致,下面是一些创建数组的实例：

```
int arraylnts[6];
int arrayNums[] = {2,4,6,8,11};
int arrayVals[6] = {2,4,-8,3,5,7};
char arrayString[7] = "Anlaino" ;
```

由实例可以看出,Arduino 数组的创建可以指定初始值,如果没有指定初始值,则编译器默认为 0；同时,如果不指定数组的大小,则编译器在编译时会通过计算数据的个数来指定数组的大小。

数组被创建之后,可以指定数组中某个数据的值：

```
int intArray[5];
intArray[2] = 2;
```

数组是从零开始索引的。也就是说,数组被初始化之后,其中第一个数据的索引为 0,如上例所示,arrayVals[0]＝2,0 为数组第一个元素 2 的索引号,以此类推,在这个包含 6 个元素的数组中,5 是最后一个元素 7 的索引号,即 arrayVals[5]＝7,而 arrayVals[6]是无效的,它将会是任意的随机信息(内存地址)。

程序 3-2 先创建了一个数组,然后在 for 循环中,将数组中的每一个数据送到串口打印。

```
1   /**********************************************************
2    * 程序 3-2: 数组的使用
3    **********************************************************/
4   void setup()
5   {
6     //定义长度为 10 的数组
7     int intArray[10] = {1,2,3,4,5,6,7,8,9,10};
8     int i;
9     for(i = 0; i < 10; i = i + 1)             //循环遍历数组
10    {
11      Serial.println(intArray[i]);            //打印数组元素
12    }
13  }
14  void loop()
15  {
16  }
```

(7) 字符串。字符串的定义方式有两种：一种是以字符型数组的方式定义的，另一种是用 String 类型定义的。

以字符型数组方式定义的语句为：

```
char 字符串名称[字符个数];
```

以字符型数组方式定义字符串的使用方法与数组的使用方法一致，有多少个字符就占用多少字节的存储空间。

在大多数情况下都使用 String 类型来定义字符串。该类型提供了一些操作字符串的成员函数，使字符串使用起来更为灵活。其定义语句为：

```
String 字符串名称;
```

字符串既可以在定义时赋值，也可以在定义以后赋值。假设定义一个名为 str 的字符串，则下面两种方式是等效的：

```
String str;
str = "Arduino";
```

或者

```
String str = "Arduino";
```

3.2.3 数据运算

最常用的 Arduino 运算符包括赋值运算符、算术运算符、关系运算符、逻辑运算符及递增/减运算符。

(1) 赋值运算符。

＝(等于)为指定某个变量的值,如 A＝x,将变量 x 的值放入变量 A 中。

＋＝(加等于)为加上某个变量的值,如 B＋＝x,将变量 B 的值与变量 x 的值相加,将二者的和放入变量 B 中,与 B＝B＋x 表达式相同。

－＝(减等于)为减去某个变量的值,如 C－＝x,将变量 C 的值减去变量 x 的值,差放入变量 C 中,与 C＝C－x 表达式相同。

＝(乘等于)为乘以某个变量的值,如 D＝x,将变量 D 的值与变量 x 的值相乘,积放入变量 D 中,与 D＝D*x 表达式相同。

/＝(除等于)为除以某个变量的值,如 E/＝x,将变量 E 的值除以变量 x 的值,商放入变量 E 中,与 E＝E/x 表达式相同。

％＝(取余等于)为对某个变量的值取余数,如 F％＝x,将变量 F 的值除以变量 x 的值,余数放入变量 F 中,与 F＝F％x 表达式相同。

&＝(与等于)为对某个变量的值按位进行与运算,如 G&＝x,将变量 G 的值与变量 x 的值进行 AND 运算,结果放入变量 G 中,与 G＝G&x 表达式相同。

|＝(或等于)为对某个变量的值按位进行或运算,如 H|＝x,将变量 H 的值与变量 x 的值进行 OR 运算,结果放入变量 H 中,与 H＝H|x 表达式相同。

^＝(异或等于)为对某个变量的值按位进行异或运算,如 I^＝x,将变量 I 的值与变量 x 的值进行 XOR 运算,结果放入变量 I 中,与 I＝I^x 表达式相同。

<<＝(左移等于)为将某个变量的值按位进行左移,如 J <<＝n,将变量 J 的值左移 n 位,与 J＝J << n 表达式相同。

>>＝(右移等于)为将某个变量的值按位进行右移,如 K >>＝n,将变量 K 的值右移 n 位,与 K＝K >> n 表达式相同。

(2) 算术运算符。

＋(加)为对两个值求和,如 A＝x＋y,将变量 x 与 y 的值相加,和放入变量 A 中。

－(减)为对两个值相减,如 B＝x－y,将变量 x 的值减去变量 y 的值,差放入变量 B 中。

*(乘)为对两个值相乘,如 C＝x*y,将变量 x 与 y 的值相乘,积放入变量 C 中。

/(除)为对两个值相除,如 D＝x/y,将变量 x 的值除以变量 y 的值,商放入变量 D 中。

％(取余)为对两个值做取余运算,如 E＝x％y,将变量 x 的值除以变量 y 的值,余数放入变量 E 中。

(3) 关系运算符。

＝＝(相等)为相等关系运算符, 如 x＝＝y,若变量 x 与 y 的值相等,则其结果为 1,不相等则其结果为 0。

!＝(不相等)为不相等关系运算符,如 x!＝y,若变量 x 与 y 的值不相等,则其结果为 1,相等则其结果为 0。

<(小于)为小于关系运算符,如 x < y,若变量 x 的值小于变量 y 的值,则其结果为 1,否则其结果为 0。

>(大于)为大于关系运算符,如 x>y,若变量 x 的值大于变量 y 的值,则其结果为 1,否则其结果为 0。

<=(小于或等于)为小于或等于关系运算符,如 x<=y,若变量 x 的值小于或等于变量 y 的值,则其结果为 1,否则其结果为 0。

>=(大于或等于)为大于或等于关系运算符,如 x>=y,若变量 x 的值大于或等于变量 y 的值,则其结果为 1,否则其结果为 0。

(4) 逻辑运算符。

&&(与运算)为对两个表达式的布尔值进行按位与运算,如(x>y)&&(x<z),若变量 x 的值大于变量 y 的值并且若变量 x 的值小于变量 z 的值,则其结果为 1,否则其结果为 0。

||(或运算)为对两个表达式的布尔值进行按位或运算,如(x>y)||(x<z),若变量 x 的值大于变量 y 的值或者变量 x 的值小于变量 z 的值,则其结果为 1,否则其结果为 0。

!(非运算)为对某个布尔值进行非运算,如!(x>y),若变量 x 的值大于变量 y 的值,则其结果为 0,否则其结果为 1。

(5) 递增/减运算符。

++(加 1)为将运算符左边的值自动增 1,如 x++,将变量 x 的值加 1,表示在 x 使用后,再使 x 值加 1。

--(减 1)为将运算符左边的值自动减 1,如 x--,将变量 x 的值减 1,表示在使用 x 后,再使 x 值减 1。

3.3 程序结构

任何复杂的算法都可以由顺序结构、循环结构及选择结构这 3 种基本结构组成,在构造算法时也仅以这 3 种结构作为基本单元。一个复杂的程序可以被分解为若干个结构和若干层子结构,从而使程序结构的层次分明、清晰易懂,易于进行正确性的验证和纠正程序中的错误。

3.3.1 顺序结构

在 3 种程序结构中,顺序结构是最基本、最简单的程序组织结构。在顺序结构中,程序按语句的先后顺序依次执行。一个程序或者一个函数在整体上是一个顺序结构,由一系列语句或者控制结构组成。这些语句与控制结构都按先后顺序运行。

程序 3-3 中的 loop()函数的 4 条语句就是顺序执行的。

```
1  /*********************************************************
2   * 程序 3-3: 顺序结构
3   *********************************************************/
4  int ledPin = 13;
```

```
5   int delayTime = 1000;
6   void setup()
7   {
8     pinMode(ledPin,OUTPUT);
9   }
10  void loop()
11  {
12    digitalWrite(ledPin,HIGH);          //点亮 LED
13    delay(delayTime);                   //延时
14    digitalWrite(ledPin,LOW);           //熄灭 LED
15    delay(delayTime);                   //延时
16  }
```

delay()函数使程序暂停设定的时间(ms),例如,delay(500)即为延时 500ms。

3.3.2 选择结构

选择结构又称为选取结构或分支结构。编程过程经常需要根据当前的数据做出判断后,再进行不同的选择。这时就会用到选择结构,即针对同一个变量,根据不同的值,程序执行不同的语句。

选择语句有以下两种形式。

(1) if 语句。

if 语句是最常用的选择结构实现方式,当给定的表达式为真时,就会执行其后的语句。if 语句有 3 种结构形式。

第一种是简单分支结构,语法结构为:

```
if(表达式)
{
    语句;
}
```

程序 3-4 通过改变延时时间来控制小灯由慢到快进行闪烁,到达一定的频率后恢复初始频率。

```
1   /*********************************************************
2    * 程序 3-4: if 语句的使用
3    *********************************************************/
4   int ledPin = 13;
5   int delayTime = 1000;
6   void setup()
7   {
8     pinMode(ledPin,OUTPUT);
9   }
10  void loop()
```

```
11  {
12    digitalWrite(ledPin,HIGH);                    //点亮小灯
13    delay(delayTime);                             //延时
14    digitalWrite(ledPin,LOW);                     //熄灭小灯
15    delay(delayTime);
16    delayTime = delayTime - 100;                  //每次将延时时间减少 0.1s
17    if(delayTime < 100)
18    {
19      delayTime = 1000;                           //当延时时间少于 0.1s 时,重设为 1s
20    }
21  }
```

该程序用到了 if 条件判断语句，每次运行到 if 语句时都会进行判断，在 delayTime >= 100 时，大括号里面的 delayTime=1000 是不执行的，进入下一次循环；当 delayTime < 100 时，delayTime=1000 被执行，delayTime 的值变为 1000，进入下一次循环。

第二种是双分支结构，语法结构为：

```
if(表达式)
{
    语句 1;
}
else
{
    语句 2;
}
```

该结构增加了一个 else 语句，当给定表达式的结果为假时，便会运行 else 后的语句。程序 3-5 与程序 3-4 实现的功能相同，但利用了双分支结构。

```
1   /**********************************************************
2   * 程序 3 - 5: 双分支结构 if 语句的使用
3   **********************************************************/
4   int ledPin = 13;
5   int delayTime = 1000;
6   void setup()
7   {
8     pinMode(ledPin,OUTPUT);
9   }
10  void loop()
11  {
12    digitalWrite(ledPin,HIGH);                      //点亮小灯
13    delay(delayTime);                               //延时
14    digitalWrite(ledPin,LOW);                       //熄灭小灯
15    delay(delayTime);
```

```
16    if(delayTime < 100)
17    {
18      delayTime = 1000; //当延时时间少于 0.1s 时,重设为 1s
19    }
20    else
21    {
22      delayTime = delayTime - 100; //每次将延时时间减少 0.1s
23    }
24  }
```

第三种结构为多分支结构,可以判断多种情况,语法结构为:

```
if(表达式 1)
{
    语句 1;
}
else if(表达式 2)
{
    语句 2;
}
else if(表达式 3)
{
     语句 3;
}
else
{
      语句 4;
}
...
```

程序 3-6 可使小灯的闪烁频率在不同的时间段内保持一定的闪烁频率,达到一定的时间后,重新恢复初始的闪烁频率。

```
1   /***********************************************************
2    * 程序 3-6: 多分支结构 if 语句的使用
3    ***********************************************************/
4   int ledPin = 13;
5   int delayTime = 1000;
6   void setup()
7   {
8     pinMode(ledPin,OUTPUT);
9   }
10  void loop()
11  {
```

```
12    digitalWrite(ledPin,HIGH);                    //点亮小灯
13    delay(delayTime);                             //延时
14    digitalWrite(ledPin,LOW);                     //熄灭小灯
15    delay(delayTime);
16    if(delayTime > 800 && delayTime <= 1000)
17    {
18      delayTime = delayTime - 100;                //每次将延时时间减少 0.1s
19    }
20    else if(delayTime > 500 && delayTime <= 800)
21    {
22      delayTime = delayTime - 50;                 //每次将延时时间减少 0.05s
23    }
24    else if(delayTime > 200 && delayTime <= 500)
25    {
26      delayTime = delayTime - 20;                 //每次将延时时间减少 0.02s
27    }
28    else
29    {
30      delayTime = 1000;                           //将延时时间重新设定为 1s
31    }
32  }
```

（2）switch…case 语句。

处理比较复杂的问题时，可能会存在很多选择分支的情况，如果还使用 if 的结构编写程序，则会使程序冗长，可读性差，此时可以使用 switch…case 语句。switch…case 语句的语法结构为：

```
switch(表达式 1)
{
    case 常量表达式 1:
      语句 1;
      break;
    case 常量表达式 2:
      语句 2;
      break;
    case 常量表达式 3:
      语句 3;
      break;
    ...
    default:
      语句 n;
      break;
}
```

switch 结构会将 switch 语句后的表达式与 case 后的常量表达式进行比较，如果相符，

则运行常量表达式所对应的语句；如果不相符，则会运行 default 后的语句。

程序 3-7 判定一个给定值，当给定值为 1 时，红灯闪烁；当给定值为 2 时，绿灯闪烁；当给定值为 3 时，蓝灯闪烁。

```
1   /************************************************************
2    * 程序 3-7：多分支结构 switch 语句的使用
3    ************************************************************/
4   int ledRed = 5;
5   int ledGreen = 6;
6   int ledBlue = 7;
7   int num = 1;               //可以改变该值为 2、3，观察不同灯的闪烁情况
8   void setup()
9   {
10    pinMode(ledRed,OUTPUT);
11    pinMode(ledGreen,OUTPUT);
12    pinMode(ledBlue,OUTPUT);
13  }
14  void loop()
15  {
16    switch(num)
17    {
18    case 1:
19      digitalWrite(ledRed,HIGH);
20      delay(200);
21      digitalWrite(ledRed,LOW);
22      break;
23    case 2:
24      digitalWrite(ledGreen,HIGH);
25      delay(200);
26      digitalWrite(ledGreen,LOW);
27      break;
28    case 3:
29      digitalWrite(ledBlue,HIGH);
30      delay(200);
31      digitalWrite(ledBlue,LOW);
32      break;
33    }
34  }
```

3.3.3 循环结构

(1) for 循环语句。

for 循环语句可以控制循环的次数，语法结构为：

```
for(初始化; 条件检测; 循环状态)
```

```
{
    程序语句;
}
```

初始化语句是对变量进行条件初始化。条件检测语句是对变量的值进行条件判断。如果为真,则运行在 for 循环语句大括号中的内容;如果为假,则跳出循环。执行完大括号中的语句后,接着执行循环状态语句,然后重新执行条件检测语句。

程序 3-8 利用 for 循环,让小灯每闪烁 20 次,就暂停 3s。

```
1   /*********************************************************
2    * 程序 3-8: for 循环语句的使用
3    *********************************************************/
4   int ledPin = 13;
5   int delayTime = 100;                  //定义小灯闪烁间隔 delayTime 为 0.1s
6   int count;                            //定义计数器变量
7   void setup()
8   {
9     pinMode(ledPin,OUTPUT);
10  }
11  void loop()
12  {
13    for(count = 0; count < 20; count++)
14    {
15      digitalWrite(ledPin,HIGH);
16      delay(delayTime);
17      digitalWrite(ledPin,LOW);
18      delay(delayTime);
19    }
20    delay(3000);                        //延时 3s
21  }
```

(2) while 循环语句。

while 循环语句的语法为:

```
while(条件语句)
{
  程序语句
}
```

当条件语句为真时,则执行循环中的程序语句;否则跳出 while 循环,执行后续语句。

程序 3-9 利用 while 循环,完成与程序 3-8 相同的功能(小灯每闪烁 20 次就暂停 3s)。

```
1   /*********************************************************
2    * 程序 3-9: while 循环语句的使用
3    *********************************************************/
```

```
4   int ledPin = 13;
5   int delayTime = 100;                         //定义小灯闪烁间隔 delayTime 为 0.1s
6   int count;                                   //定义计数器变量
7   void setup()
8   {
9     pinMode(ledPin,OUTPUT);
10  }
11  void loop()
12  {
13    count = 0;
14    while(count < 20)
15    {
16      digitalWrite(ledPin,HIGH);
17      delay(delayTime);
18      digitalWrite(ledPin,LOW);
19      delay(delayTime);
20      count ++;
21    }
22    delay(3000);                               //延时 3s
23  }
```

程序 3-8 与程序 3-9 的缺点是：虽然可以在一个 loop()函数中，通过 for 或 while 循环来完成闪灯 20 次后延时 3s 的要求，但是 loop()函数的执行时间过长。而在应用中，经常会在 loop()函数中检查是否有中断或者其他信号，如果单次 loop()执行时间较长，那么不可避免地会增加程序的响应时间。因此，比较而言，采用 if 语句和 count 计数器更好些。

程序清单 3-10 如下：

```
1   /*********************************************************
2    * 程序 3-10：使用 if 语句和 count 计数器的闪灯程序
3    *********************************************************/
4   int ledPin = 13;
5   int delayTime = 1000;                        //定义延时变量 delayTime 为 1s
6   int delayTime2 = 3000;                       //定义延时变量 delayTime2 为 3s
7   int count = 0;                               //定义计数器变量并初始化为 0
8   void setup()
9   {
10      pinMode(ledPin,OUTPUT);
11  }
12  void loop()
13  {
14      digitalWrite(ledPin,HIGH);
15      delay(delayTime);
16      digitalWrite(ledPin,LOW);
17      delay(delayTime);
```

```
18      count ++;
19      if(count == 20)
20      {
21          delay(delayTime2); //当计数器数值为 20 时,延时 3s
22          count = 0;
23      }
24  }
```

3.4 函数的使用

3.4.1 自己封装函数

以闪灯为例,LED 灯要闪烁 20 次,可以将闪灯这个功能封装到一个函数中,当多次需要闪灯时,便可以直接调用这个闪灯函数了。

```
1   /*******************************************************
2    * 程序 3-11: 自己封装函数
3    *******************************************************/
4   int ledPin = 13;
5   int delayTime = 1000;                   //定义延时变量 delayTime 为 1s
6   int delayTime2 = 3000;                  //定义延时变量 delayTime2 为 3s
7   int count;
8   void setup()
9   {
10    pinMode(ledPin,OUTPUT);
11  }
12  void loop()
13  {
14    for(count = 0; count < 20; count ++)    //调用 20 次闪烁函数
15    {
16      flash();
17    }
18    delay(delayTime2);                    //延时 3s
19  }
20  void flash()                            //定义无参数的闪灯函数
21  {
22    digitalWrite(ledPin,HIGH);
23    delay(delayTime);
24    digitalWrite(ledPin,LOW);
25    delay(delayTime);
26  }
```

在该程序里,调用的 flash()函数实际上就是 LED 闪烁的代码,相当于程序运行到这个

函数时,便跳入到4行闪灯代码中,函数非常简单。

有些函数需要接收参数才能执行特定的功能。在这个例子中,flash()函数很简单,它没有任何返回值,而且也没有任何参数。

3.4.2 函数中的参数传递

所谓函数参数,就是函数中需要传递值的变量、常量、表达式、函数等。接下来的例子会将闪灯函数改造一下,使其闪烁时间可以变化。

```
1   /*********************************************************
2    * 程序 3-12: 函数中的参数传递
3    *********************************************************/
4   int ledPin = 13;
5   int delayTime = 1000;                          //定义延时变量 delayTime 为 1s
6   int delayTime2 = 3000;                         //定义延时变量 delayTime2 为 3s
7   int count;
8   void setup()
9   {
10    pinMode(ledPin,OUTPUT);
11  }
12  void loop()
13  {
14    for(count = 0; count < 20; count++)
15    {
16      flash(delayTime);                          //调用 20 次闪烁灯光的函数,延时为 1s
17    }
18    delay(delayTime2);                           //延时 3s
19  }
20  void flash(int delayTime3)                     //定义具有参数的闪灯函数
21  {
22    digitalWrite(ledPin,HIGH);
23    delay(delayTime3);
24    digitalWrite(ledPin,LOW);
25    delay(delayTime3);
26  }
```

在改进的闪灯例子中,flash()函数接收一个整型的参数 delayTime3,称为形参,全名为形式参数。形参是在定义函数名和函数体时使用的参数,目的是用来接收调用该函数时传递的参数,值一般不确定。形参变量只有在被调用时才分配内存单元,在调用结束时,即刻释放所分配的内存单元。因此,形参只在函数内部有效。函数调用结束并返回主调用函数后,则不能再使用该形参变量。

loop()函数中 flash()接收的参数 delayTime 称为实参,全名为实际参数。实参是传递给形参的值,具有确定的值。实参和形参在数量上、类型上、顺序上应严格一致,否则将会发

生类型不匹配的错误。

3.4.3　非空类型的函数

如果是非空类型的函数，那么在构造函数时应注意函数的返回值应和函数的类型保持一致，在调用该函数时，函数返回值应和变量的类型保持一致。

下面的程序中包含一个比较两个整数大小的函数 Max()，并给出了这个具有返回值的函数的定义和调用。

```
1   /*******************************************************
2    * 程序 3-13: 非空类型的函数
3    *******************************************************/
4   //定义求两个数最大值的函数
5   int Max(int a,int b)
6   {
7     if(a>=b)
8     {
9       return a;                    //a>=b 时返回 a
10    }
11    else
12    {
13      return b;                    //a<b 时返回 b
14    }
15  }
16  void setup()
17  {
18    int x=Max(10,20);              //调用 Max()函数
19    Serial.println(x);
20  }
21  void loop()
22  {
23  }
```