第 5 章

流程控制语句

在一个程序执行的过程中，各条语句的执行顺序对程序的结果是有直接影响的。控制语句用实现对程序流程的选择、循环、转向和返回等进行控制。只有在清楚每条语句的执行流程的前提下，才能通过控制语句的执行顺序实现要完成的任务。

Dart 中的控制语句分为以下几类：

(1) 分支语句：if 和 switch。

(2) 循环语句：while、do-while 和 for。

(3) 跳转语句：break、continue、return 和 throw。

创建命令行应用程序，将项目命名为 chapter5。本章所有文件都在该项目的 bin 文件夹下创建与运行。

11min

5.1 分支语句

分支语句又称条件语句，条件语句使得部分程序块根据表达式的值有选择地执行。Dart 语言提供了 if 和 switch 两种分支语句。

5.1.1 if 语句

具体来说分支语句由选择结构组成，if 语句引导的选择结构包括 if 结构、if-else 结构和 else-if 结构。

1. if 结构

if 结构包含条件表达式和语句块，当条件表达式为 true 时执行语句块，否则执行 if 结构后面的语句。通常来说语句块由大括号({})包裹，如果 if 结构中的语句块只有一条语句，则可以省略大括号。

if 结构声明语法如下：

```
if(条件表达式){
  //语句块
}
```

示例代码如下：

```
//chapter5/bin/example_01.dart
void main(){
  var x = true,y = 10;
  if(x){
    //此处 x 为真,语句体得到执行
    print('条件 x 为 $ x');
  }
  if(!x){
    //此处!x 为假,语句体得不到执行
    print('条件!x 为 $ x,if 语句体得到执行.');
  }
  //此处省略大括号
  if(y<=10)
    //条件 y<=10 为真,语句体得到执行
    print('条件 y<=10 为 $ {y<=10}');
}
```

2. if-else 结构

在 if 结构中，只有条件表达式为 true 时提供相应的处理语句块。有时无论条件表达式为 true 还是 false 都需进行相应处理，此时可以使用 if-else 结构。声明格式如下：

```
if(条件表达式){
  //语句块 1
}else{
  //语句块 2
}
```

当程序执行到 if 语句时，先判断条件表达式，如果值为 true，则执行语句块 1，然后跳过 else 语句块，继续执行后续语句。如果条件表达式值为 false，则忽略语句块 1 直接执行语句块 2，然后继续执行后续语句。

if-else 结构使用示例代码如下：

```
//chapter5/bin/example_02.dart
void main(){
  var x = true;
  if(!x){
    //条件!x 为真时,if 语句块得到执行
    print('if !x: $ {!x}');
  }else{
    //条件!x 为假时,else 语句块得到执行
    print('else !x: $ {!x}');
  }
}
```

3. else-if 结构

else-if 结构是 if-else 结构的多层嵌套形式，它会在多个分支中执行一个语句块，其他分支将不会得到执行，所以这种结构常用于有多种判断结果的分支中。

else-if 结构如下：

```
if(条件表达式 1){
  //语句块 1
}else if(条件表达式 2){
  //语句块 2
}
…
}else if(条件表达式 n){
  //语句块 n
}else{
  //语句块 n + 1
}
```

else-if 结构使用示例代码如下：

```
//chapter5/bin/example_03.dart
void main(){
  var x = true,y = 10;
  if(!x){
    //条件!x 为假, if 语句块不会执行
    print('条件 x 为 $ x');
  }else if(y < = 10){
    //如果条件 y < = 10 为真,则 else if 语句块得到执行
    print('条件 y < = 10 为 $ {y < = 10}');
  }else{
    //当所有条件都为假时,else 语句块才会执行
    print('else !x: $ {!x}.');
  }
}
```

5.1.2 switch 语句

可以使用 switch 语句选择要执行的多个代码块中的一个代码块，语法如下：

```
switch(expr){
  case 值 1:
    //语句块 1
    break;
  case 值 2:
    //语句块 2
```

```
    break;
    …
  case 值 n:
    //语句块 n
    break;
  default:
    //语句块 n + 1
}
```

首先设置表达式 expr，通常是一个变量。随后表达式的值会与结构中的每个 case 的值做比较。如果存在匹配，则与该 case 子句关联的代码块会被执行。如果 case 子句中没有与 expr 匹配的值，则会执行 default 子句的语句块，然后跳出 switch 结构。

switch 语句使用＝＝来比较整数、字符串或编译时常量，比较的两个对象必须是同一个类的实例且没有覆写＝＝运算符。

每个非空的 case 子句必须包含一个 break 语句。当没有匹配的子句时，可以使用 default 子句匹配这种情况。示例代码如下：

```
//chapter5/bin/example_04.dart
void main(){
  var color = 'Red';
  switch(color){
    case 'Green':
      print('color 值为 Green');
      break;
    case 'Orange':
      print('color 值为 Orange');
      break;
    case 'Red':
      print('color 值为 Red');
      break;
    default:
      print('color 值未匹配');
  }
}
```

下面的例子忽略了 break 子句，因此产生错误。

```
//chapter5/bin/example_05.dart
void main(){
  var color = 'Red';
  switch(color){
    case 'Green':
      print('color 值为 Green');
```

```
  //执行出错,没有 break 语句
    case 'Orange':
      print('color 值为 Orange');
      break;
    case 'Red':
      print('color 值为 Red');
      break;
    default:
      print('color 值未匹配');
  }
}
```

Dart 语言支持 case 子句的内容为空,这种情况称为贯穿。下例中值为 Orange 的 case 子句内容为空,它会造成贯穿。当传入变量值为 Orange 时,程序会继续执行与它相邻的值为 Red 的 case 子句。示例代码如下:

```
//chapter5/bin/example_06.dart
void main(){
  var color = 'Orange';
  switch(color){
    case 'Green':
      print('color 值为 Green');
      break;
    case 'Orange':
      //空语句贯穿
    case 'Red':
      //color 值为 Orange 和 Red 都会被执行
      print('color 值为 Red');
      break;
    default:
      print('color 值未匹配');
  }
}
```

非空 case 子句实现贯穿可以通过 continue 语句和标签来完成。下例中匹配到值为 Green 的 case 子句时,不仅会执行该子句的代码块,还会执行值为 Red 的 case 子句中的代码块。示例代码如下:

```
//chapter5/bin/example_07.dart
void main(){
  var color = 'Green';
  switch(color){
    case 'Green':
      print('color 值为 Green');
```

```
        //继续执行标签为 red 的 case 子句
        continue red;
      case 'Orange':
        print('color 值为 Orange');
        break;
      //标签 red,它代表了 case 值为 Red 的子句
      red:
      case 'Red':
        //color 值为 Green 和 Red 都会被执行
        print('color 值为 Red');
        break;
      default:
        print('color 值未匹配');
    }
}
```

每个 case 子句都可以有局部变量且仅在该 case 子句内有效。

5.2　循环语句

9min

循环语句能够根据循环条件使程序代码重复执行。Dart 支持 3 种循环结构：for、while 和 do-while。

5.2.1　for 语句

for 语句是一种应用广泛、功能最强的循环语句。声明格式如下：

```
for(表达式 1;表达式 2;表达式 3){
  //循环语句块
}
```

表达式 1 是初始化语句，用于初始化循环变量和其他变量。表达式 2 是循环条件，表达式 3 用于更新循环变量，使循环变量趋近于某个值，直到使循环条件变为 false。

开始 for 循环时，会执行表达式 1 初始化循环变量，表达式 1 只会执行一次。然后执行表达式 2，判断循环条件是否满足，如果满足，则继续执行循环体。执行完成后继续执行表达式 3，更新循环变量，然后接着再判断循环条件。如此反复，直到循环条件不满足时跳出循环。执行流程如图 5-1 所示。

for 循环使用示例代码如下：

```
//chapter5/bin/example_08.dart
void main(){
  for (var i = 0;i < 5;i++){
```

```
    //此处 i 的初始值为 0,所以使用 i + 1 保证其与实际循环次数一致
    print('第 ${i + 1}次循环.');
  }
}
```

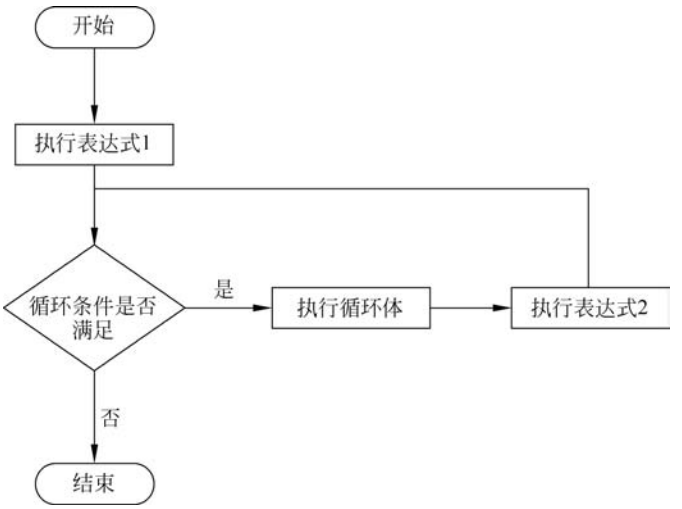

图 5-1　for 循环执行流程图

循环开始时,给循环变量 i 赋值为 0,每次执行循环体前都会判断 i 的值是否小于 5,如果为 true,则执行循环体,然后执行 i++,使得循环变量 i 的值加 1。然后继续判断循环条件,直到判断结果为 false 时跳出循环。

对于实现了 iterable 接口的类可以使用 forEach()函数进行迭代。例如 List 和 Set 类,它们还支持 for-in 形式的迭代。示例代码如下:

```
//chapter5/bin/example_09.dart
void main(){
  var collection = [0, 1, 2];
  //这里给 forEach 函数传入了一个匿名函数
  collection.forEach((i) => print(i));
  //for - in 形式的迭代
  for (var x in collection) {
    print(x);
  }
}
```

在 Dart 语言中,for 循环中的闭包会自动捕获索引值,这在某些情况下很有用。示例代码如下:

```
//chapter5/bin/example_10.dart
void main(){
  var callbacks = [];
  for (var i = 0; i < 2; i++) {
    callbacks.add(() => print(i));
  }
  callbacks.forEach((c) => c());
}
```

5.2.2　while 语句

while 循环在执行循环体前先判断循环条件，如果条件为真则执行循环体，如果条件为假则结束循环。

声明格式如下：

```
while(循环条件){
    //语句块
  }
```

在 while 循环体中一定要存在影响循环条件的语句，其作用是使循环条件最终变为 false 以避免无限循环。

示例代码如下：

```
//chapter5/bin/example_11.dart
void main(){
  var i = 0,x = 10;
  while(i < x){
    print('第 ${i + 1}次 while 循环.');
    //影响条件表达式并使其趋于假的语句
    i++;
  }
}
```

示例中 i++是影响循环条件的因子，也是它使得循环能够在有限次循环后结束。

5.2.3　do-while 语句

do-while 循环与 while 循环类似。声明格式如下：

```
do
    //语句块
}while(循环条件){
```

不同之处在于 do-while 循环会先执行一次循环体，再判断循环条件，如果条件为真则执行循环体，如果条件为假则结束循环。

示例代码如下：

```
//chapter5/bin/example_12.dart
void main(){
  var i = 0,x = 10;
  do{
    print('第 ${i + 1}次 do while 循环.');
    //影响条件并使其趋于假的语句
    i++;
  }while(i < x);
}
```

6min

5.3 跳转语句

跳转语句可以改变代码的执行顺序，从而可以实现跳转。

5.3.1 break 语句

break 语句可用于 for、while 和 do-while 循环结构，它的作用是强行退出循环体，且不再执行循环体中剩余的语句。

在 while 循环中 break 语句使用示例代码如下：

```
//chapter5/bin/example_13.dart
void main(){
  var i = 0,x = 10;
  while(i < x){
    print('第 ${i + 1}次 while 循环.');
    if(i == 5){
      //当循环变量 i 等于 5 时跳出 for 循环
      break;
    }
    //影响条件表达式并使其趋于假的语句
    i++;
  }
}
```

5.3.2 continue 语句

continue 语句的作用是结束本次循环，跳过循环体中尚未执行的语句，接着继续判断循环条件，以决定是否继续循环。

在 for 循环中 continue 语句使用示例代码如下：

```
//chapter5/bin/example_14.dart
void main(){
  for (var i = 0; i < 10; i++) {
    if(i == 5) continue;
    //此处 i 的初始值为 0,所以使用 i + 1 保证其与实际循环次数一致
    print('第 ${i + 1}次循环.');
  }
}
```

5.3.3　assert

assert 函数又称断言，它的第一个参数是布尔表达式，在表达式的值为 false 时中断代码执行，断言在开发过程中用于检测对象是否符合期待。示例代码如下：

```
//chapter5/bin/example_15.dart
void main(){
  var x,y = 9;
  //确保变量 x 的值非空
  assert(x != null);
  //确保变量 x 的值小于 8
  assert(y < 8);
}
```

也可以向 assert 函数传递一个 String 类型的可选参数。示例代码如下：

```
//chapter5/bin/example_16.dart
void main(){
  var URLString = 'http://dartlang.org';
  //确保网址应该以 https 开始
  assert(URLString.startsWith('https'));
}
```

断言什么时候起作用取决于使用的工具和框架：

(1) Flutter 在调试模式下启用断言。

(2) 默认情况下，仅开发工具(例如 dartdevc)通常启用断言。

(3) 某些工具(例如 dart 和 dart2js)通过命令行标志---enable-asserts 支持断言。

(4) 在生产代码中，断言将被忽略，并且不会评估断言的参数。