

特征	描述	示例
命名空间	引用模型元素时，您必须对这些元素添加前缀。存在下列前缀： dp: (表示数据源项)， ev: (表示事件)， v: (表示局部变量)， f: (表示函数)	<pre>dp:x = 100; // 设置数据源项 fire ev:back(); // 触发事件 f:trace_string("hello world"); // 调用函数</pre>
访问数据源项	通过将数据源项放置在赋值左侧来编写此项。通过在表达式中的任何其他位置使用数据源项来读取此项。重定向引用 (=>) 是数据源项赋值的特殊形式。	<pre>dp:x = 5; // 写入 x dp:x = dp:y + dp:z; // 读取 y 和 z length dp:aList; // 读取列表数据源项的长度 dp:refX => dp:x; // 重定向</pre>
发送事件	语法： fire ev: <identifier>(<parameter-list>); 可以在超时后发送事件。 可以使用 cancel_fire 表达式取消此延迟事件。 语法： fire_delayed <timeout>, ev: <identifier>(<parameter-list>); cancel_fire ev: <identifier>;	<pre>fire ev:back(); fire ev:mouseClick(10, 20); fire_delayed 3000, ev:back(); // 在 3 秒内发送时间“返回”。 cancel_fire ev:back; // 取消事件</pre>
对事件做出响应	要对事件做出响应，请使用 match_event 。这是 if-then-else 语句的特殊形式。 if 和 else 分支必须始终具有相同类型。如果在赋值右侧使用，则 else 分支是必需的。 语法： match_event v: <identifier> = ev: <identifier> in <sequence> else <sequence>	<pre>match_event v:event = ev:back in { f:trace_string("back event received"); } v:this.x = match_event v:event = ev:back in 10 else 0;</pre>
访问事件参数	match_event 的 in 表达式可以访问事件参数。使用点标记来访问事件参数。	<pre>match_event v:event = ev:mouseClick in { v:this.x = v:event.x; v:this.y = v:event.y; }</pre>
访问控件属性	如果某个脚本是小组件(小组件操作、输入响应)的组成部分，则它可以访问该小组件的属性。通过引用当前小组件的变量，名为 v:this 的特殊局部变量可用。使用点标记对控件属性进行寻址。	<pre>v:this.text = "hello world"; v:this.x = 10;</pre>
浏览小组件树	如果某个脚本是小组件的组成部分，则它可以访问其他小组件的属性。使用小组件树导航运算符： -> 。要访问父代小组件，请使用标识符 ^ 。	<pre>v:this->^->caption.text = "Play"; // 转到父代，转至标题，属性 text v:this->^..x = 1; // 转到父代，属性 x</pre>
字符串格式设置	+ 运算符用于连接字符串。有关更多字符串转换函数，请参阅文档	<pre>v:this.text = "current speed:" + f:int2string(dp:speed) + "km/h";</pre>

特征	描述	示例
常量	编写的字符串常量可以不带引号。 颜色常量是 RGBA 四重。	<pre> “hello world” // 字符串常量 Napoleon // 字符串常量 5 // 整数常量 color:0,235,0,255 // EB 绿色 </pre>
算术、逻辑和赋值运算符	加法和字符串连接:+, 减法: -, 乘法: *, 除法: /, 取模: %, 大于: >, 小于: <, 大于或等于: >=, 小于或等于: <=, 等于: ==, 不等于: !=, 与: &&, 或: , 非: !, 赋值: =, 赋值增加: +=, 赋值减小: -=	<pre> dp:myString = “Hello” + “World”; dp:count += 1; // 增加一 </pre>
序列	序列可以是单个表达式, 也可以括在大括号中的一系列表达式。序列中的最后一个表达式为序列的值。	<pre> if(dp:something) dp:x = 5; // 单个表达式 if(dp:other) { dp:x = 5; // 括在大括号 dp:y = 10; // 中的表达式 } </pre>
局部变量	使用 let 绑定引入局部变量。不允许使用未初始化的变量。 let 绑定可以进行嵌套。 语法: let v :<identifier> = <expression>; v :<identifier2> = <expression>; ... in <sequence>	<pre> let v:x = 42; v:text = “hello world”; in { v:this.x = v:x; v:this.text = v:text; } </pre>
WHILE 循环	while 循环包含两个表达式: 条件表达式和主体表达式。在条件生成 false 之前, 将对主体重复求值。 语法: while (<expression>) <sequence>	<pre> dp:i = 0; while(dp:i <= 10) { dp:sum += i; dp:i += 1; } </pre>
IF-THEN-ELSE	If-then-else 的行为类似于 C 和 Java 中的三元条件运算符。如果在赋值右侧使用它, 则 else 分支是必需的并且这两个分支必须具有相同类型。 语法: if (<expression>) <sequence> else <sequence>	<pre> if(dp:buttonClicked) { v:this.x = dp:x; } else { v:this.x = 0; } v:this.x = if(dp:buttonClicked) dp:x else 0; </pre>
注释	允许使用 C 样式块注释和 C++ 样式行注释。	<pre> /* 此为 C 样式块注释 */ // 此为 C++ 样式行注释 </pre>
返回值	脚本中的最后一个表达式是返回值。 要强制使用无效类型的返回值, 请使用 unit 或 {}	<pre> dp:x + 2; // 返回数据源项 x + 2 </pre>