Poi 语言手册

李煜东 李源昊

1. Poi 语言简介

Poi 语言是一种非标记化、面向对象的高级编译网络编程语言，程序设计师可以通过编写这 1 种语言，完成与服务器的数据传输，以及与用户的内容交互。

Poi 语言基于.NET 平台进行开发，使用 C#语言编写编译器，目标语言基于 html，javascript，CSS，可以较容易地实现跨平台开发。

1.1 我们要做哪些工作

除了完成“使用 Poi 语言进行 Web 开发”的既定目标之外，我们也希望接触一门完整的语言设计的基本内容，更好地理解编译。因此我们的项目步骤如下：

STEP 1：从头开始设计一门新的语言，使其支持大部分高级编程语言的基本功能。

——进度：目前已经基本构造出语法树。

STEP 2：有针对性地进行若干容器类型（或其它高级类型）的设计，为面向 Web 编程提供基础。

STEP 3：进行面向对象编程的设计，对面向对象的支持程度不低于普通班 Mini Java 的设计工作。

——进度：目前该部分与 STEP 1 同步开展。

STEP 4：进行 Web 编程库设计。这部分我们计划类似于 C++的 STL，使用上面三步已经完成的 Poi 语言编译器，通过编写 Poi 语言代码的类库，提供较完整的 Web 编程支持。

1.2 语言特点

1) 有效的数据处理和保存机制
2) 支持数组、检索表等高效的动态结构
3) 提供多种内置库，同时用户易于进行扩展
4) Poi 语言是一门面向对象的网络编程语言
5) 原生支持函数式编程

2. 基础特性

2.1 语句简介

Poi 对大小写是敏感的，标识符不能以数字开始；

基本的语句包括定义和表达式。表达式可以是赋值语句，函数调用或者计算语句，定义可以是变量定义或者类定义；

每一条语句由分号(";")进行分隔，语句中的换行将被忽略；

表达式可以用括号("(", ")")组合为复合表达式；

注释包含单行注释和多行注释两种形式，单行注释由"//"开始，到一行末结束，多行注释由"/*"开始，到"*"结束，可以跨越多行；

2.2 编译与执行

编译由 Poi Compiler 执行，命令格式：

```
poicl <poi_source> [-o <outdir>] [-l <external_library>]
```

编译出的 html, javascript, CSS 文件可以直接配置在服务器端进行执行，也可以本地进行查看；

2.3 基本常量

数值型(numeric)　1, 1.2, 3.1415926, 0172(八进制数 172), 0xACF4(十六进制数 ACF4)

字符型(character)　"A" / "hello world!"

逻辑型(logical)　true / falsea

2.4 类型

2.4.1 基本类型

1) 数值类型

名称	别名	描述
(u)int8	(u)byte	8 位有(无)符号整数
(u)int16	(u)short	16 位有(无)符号整数
(u)int32	(u)int	32 位有(无)符号整数
(u)int64	(u)long	64 位有(无)符号整数
single	float	单精度浮点数
double		双精度浮点数
extended		扩展精度浮点数

2) 逻辑类型和字符类型

boolean(bool): 逻辑类型，值可以取 true 或 false

char: 字符类型，支持基本的 ASCII 字符

3) 映射类型

function: 函数类型，也即映射类型。Poi 语言将函数视作一个从一个 pair 到另一个 pair 一个映射；

与大多数的面向过程和面向对象的语言不同，Poi 语言将函数也视作一种类型，并可以作为变量进行赋值，传递等操作；

调用过程与大多数语言相似，使用 func(<parameter_pair>)进行调用，而返回值存储在一个 pair 中，可以同时返回多个值；

2.4.2 容器类型

具体的定义和使用方式将在下面的几节中详细描述。

1) 序对 pair

一个序对包含零或多个任何类型的变量，并可以通过变量名称引用其中的值；

2) 数组 Array

支持随机访问的一个双端队列，其中数组的长度与值类型需要显式指定；

3) 检索表 Map

存储若干<键，值>对的检索表，其中键必须为字符串类型，值可以为任何类型；

4) 事件 Event

3 定义、赋值、运算

3.1 定义与赋值

3.1.1 变量定义

Poi 语言支持对经过命名的数据进行操作，一个变量的定义遵循如下形式：

<Type> <Variable_name> [: [get [= <get_func>]], [set [= <set_func>]]] [= <initval>]

Type：变量类型，可以是内建类型或用户自定义类型

Variable_name：变量名称，要求是一个合法的标识符，且不能是保留字

访问修饰符(get，set)：可以通过设置 get，set 定义变量的访问权限和访问方式：

若声明 get{}方法，则在返回 value 前执行 get 语句块；

若声明 set{}方法，则在赋值给 value 前执行 set 语句块；

若只声明 get，set，将使用默认的赋值和取值语句；

仅声明 get 的情况下，变量将被视作一个常量；

initval：初始值，可以是变量或常量

在变量不设置初值的情况下，变量的值将被初始化为 undefined，对 undefined 值的使用将导致错误；

要注意，函数对象(function)虽然被视作对象，但有着和大多数变量不同的定义方式，function 的定义方式将在下一节中被详细描述；

3.1.2 映射类型(function)

映射类型(function)的定义遵循以下形式：

function <function_name> = <parameter_pair> -> <return_pair>
<function_stmt_block>

函数被视作从<parameter_pair>到<return_pair>的一个映射，映射规则为<function_stmt_block>；

其中<parameter_pair>和<return_pair>都可以留空，分别代表无参数和无返回值；

具体返回值的设置将在 4.3 节中进行描述；

3.1.3 赋值

Poi 语言中可以对任意的类型进行赋值，赋值语句遵循如下形式：

<Variable_name> = <value>

其中左值<Variable_name>要求必须为变量，右值<value>可以是变量或常量；

3.2 基本运算

Poi 语言支持多种一元、二元和三元操作符；这些操作符分为 14 个优先级，计算时将由优先级从高到低进行计算，若要指定计算顺序，请使用最高优先级的操作符"()"；

详细定义如下：

优先级	操作符
1(最高)	(), ., f(), a[], ++, --(后缀)
2	+, -, !, ~, ++, --(前缀), T()
3	*, /, %
4	+, -
5	<<, >>
6	<, >, <=, >=
7	==, !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14(最低)	=, +=, -=, *=, %=, <<=, >>=, &=, |=, ^=

3.2.1 函数调用，数组取值，自增与自减　优先级：1

(): 括号拥有最高的优先级，在括号中的表达式将被优先计算；

.: 用于从类等数据结构中获取对应变量；

f(): 函数调用，函数参数可以不指定或者传入一个 pair

a[]: 数组引用，获取数组中指定下标的值

++, --(后置): 后置的自增、自减运算符，只能应用于数值类型（提供运行时检测）；

3.2.2 单目运算符　　优先级：2

+, -: 单目加(减)运算符，将一个数变为正(负)形式；

!: 逻辑非

~: 位运算非

++, --(前置): 前置的自增、自减运算符，只能应用于数值类型（提供运行时检测）；

T(): 数据类型强制转换，遵循如下的转换形式：

<Type>(<Variable_name>)

3.2.3 基本算数运算　　优先级：3，4，5

*, /, %: 乘、除和取模运算符，其中取模运算符要求运算符两侧都为整数类型，除和取模运算符要求第二个操作数不为 0（提供运行时检测）；

+, -: 加、减运算符，优先级比*, /, %低一级，要求运算符两侧为相符类型（提供运

行时检测）；

<<，>>：左移，右移操作符，优先级比+，-低，要求操作数两侧都为整数类型（提供运行时检测）；

3.2.4 比较运算　　优先级：6，7

比较运算符包括<，>，<=，>=，==，!=；其中==，!=的优先级比其他运算符低；比较运算符要求运算符两侧的数据类型相符（提供运行时检测）；

3.2.5 位运算　　优先级：8，9

位运算包括&，^，|，分别代表位与，位异或和位或，并且位运算的三个运算符有严格的优先级定义：(优先级从高到低)

priority(&) > priority(^) > priority(|)

3.2.6 逻辑运算

逻辑运算包括&&和||(逻辑与和逻辑或)，其中逻辑与有着较高的优先级；

3.2.7 三目运算符

三目运算符?:，标准形式如下：

<Condition_stmt> ? <true_exp> : <false_exp>

若 Condition_stmt 为真则计算 true_exp，否则计算 false_exp

3.2.8 赋值运算

除基本的"="运算符外，Poi 语言内置+=，-=，*=，%=，<<=，>>=，&=，|=，^=运算符，这些运算符要求右值操作数遵守对应的+，-等操作符的操作数限定（见 3.2.1-3.2.5 节）

4 控制结构语句

4.1 分支语句

分支语句的标准定义如下：

```
if (<if_stmt>)
    <true_stmt_block>
[else
    <else_stmt_block>]
```

若<if_stmt>值为 true，则执行<true_stmt_block>语句块中的语句，否则执行<else_stmt_block>；

其中：

<if_stmt>必须取值逻辑真或逻辑假，或可以转换为逻辑真或逻辑假的表达式；

<true_stmt_block>和<else_stmt_block>可以包含任意的语句，包括但不限于分支、循环等语句；

4.2 循环语句

循环语句的标准定义如下：

```
for (<condition_stmt>)
init (<init_stmt_block>)
step (<step_stmt_block>)
    <loop_stmt_block>
until (condition);
```

其中：

for 语句：进入条件：每次执行循环体前判断；

init 语句：初始化语句：整个循环开始前执行；

step 语句：步进语句：每次在循环体后执行；

<loop_stmt_block>：循环体：每次条件为真时循环执行，可以包含任意语句，包括但不限于分支、循环等语句；

until 语句：停止条件：每次执行循环体后判断；

4.3 返回语句

返回语句如下：

```
return;
```

遇到 return 语句后，此函数对象的执行过程将结束，直接返回调用者；

根据 Poi 语言 function 的定义方式(3.1.2 节)，Poi 语言不在 return 语句处指定返回值，需要在 return 语句前将返回值 pair 对应变量的值进行设置；

5 容器类型

5.1 序对 pair

一个序对用如下的方式定义：

```
[<variable_declaration1>, <variable_declaration2>, <variable_declaration3>], ...]
```

其中要求每一个<variable_declaration>都是一个完整的变量定义，使用时直接使用序对中定义的变量名进行引用；

序对可以进行赋值：

```
[<variable1>, <variable2>, <variable3>, ...] = [<value1>, <value2>, <value3>, ...]
```

要求两侧的序对拥有相同的数据个数，且对应数据间可赋值（遵循赋值的基本原则，且类型对应）；

序对在函数的调用与定义中被广泛使用；

5.2 数组 Array

一个数组用如下的方式进行定义：

```
Array[<array_type>, <array_size>] <Array_name> [= <array_init>]
```

其中：

<array_type>: 数组中的元素类型；

<array_size>: 数组的大小；

<Array_name>: 数组名称

<array_init>: 可选，数组初始化器，可使用一个数组进行初始化；

Poi 语言支持多维数组的实现，可实现为：

```
Array[Array[<T>, <sizeY>], <sizeX>] <name>
```

5.3 检索表 Map

检索表支持建立“字符串”到“任意类型”的<键，值>对。其定义方式如下：

```
Map <Map_name> [= <map_init>]
```

其中：

<Map_name>: 检索表的名称

<map_init>: 初始包含的<键，值>对数据

5.4 事件 Event

表达事件的发生与对应的处理，内部实现实际上是一个包含 Array[function]和若干其它内容的 Map，用于代替 C++的函数指针和 C#的委托。当 Event 发生时 Array 中的函数被自动调用，这些函数可以使用 Map 中的其它内容。

定义方法为 Event <name>，并提供添加、去除函数和事件发生的方法（待完成）。

6 面向对象编程

6.1 基本定义

类的基本定义如下：

```
class <Class_name> {
    <variable1_declaration>
    <variable2_declaration>
    ...
}
```

由于 Poi 语言中函数也作为对象，因此类可以视作拥有多个变量；

其中类中要求提供至少一个没有返回值，输入参数任意的与类名同名的函数变量，这个函数变量将被作为构造函数在类创建时被调用；

6.2 访问修饰符

Poi 的类中的变量可以设置修饰符，方式是在定义前面加如下的修饰符之一：

public:, private:, protected:

在 public:, private:或 protected:修饰符之后，直到下一个修饰符或者类结尾的所有变量都将被设定为对应的访问权限：

public: 所有的函数变量中都可以对类中的此变量进行访问

private: 仅有此类的函数变量中可以对此变量进行访问，继承类和外部函数变量都不可访问

protected: 此类和此类的子类的函数变量中可以对此变量进行访问，外部函数变量不可访问

6.3 继承

Poi 中的类支持单继承，不支持多继承；声明单继承的方法是在<Class_name>之后加上如下语句：

extend <Extend_Class>

继承的类称作子类，被继承的类称作父类；子类将继承父类的全部 public 和 protected 变量，同时子类的函数变量中也可以对这些继承的变量进行访问；

7. Web 编程库简介

7.1 基本设计

Web 编程库是使用 Poi 语言编写的一组类库。它们将在编译时被自动与将要编译的代码打包在一起进行编译，因此 Web 编程库中的接口可以被其它代码调用。实际上，Web 编程库可以看做一个 SDK。通过 Web 编程库的设计，程序设计师可以简洁、方便地用 Poi 语言完成从静态页面布局到与服务器交互的完整的前端开发工作。Web 编程库的实现，也能验证 Poi 语言的设计是科学、全面、有生产力的。

7.2 内容介绍

Web 编程库主要包含 3 方面：

静态页面设计库：提供一些内置的类供使用者调用，以实现静态页面的设计和布局。编译器把这些类和调用代码编译成主要包含 HTML 与 CSS 的文件，以及少量 JavaScript 代码。

动态页面交互库：提供一些内置的类和方法供使用者调用，以实现页面 DOM 的动态渲染，以及用户动作的检测，以帮助开发者完成与用户的交互工作。这部分代码主要被编译为 JavaScript 代码。

动态服务器交互库：提供一些内置的类和方法供调用者调用，以实现前端页面与服务器之间的通信工作。主要包含数据传输格式、HTTP 请求与响应、AJAX 等方面的设计工作。这部分代码主要被编译为 JavaScript 代码。

8．静态页面设计库

8.1 页面布局设计

浏览器中的一个页面，在 Poi 语言中被视作一个三维的立体空间——因为页面中的各元素是可能重叠的（z-index 属性），这个三维空间用一个类的数组 Array[Page]表示。

其中 Array[]表示了 z 轴，用于实现 DOM 元素的层叠。Array 中的每个平面页面（X-Y）都是一个名为 Page 的内置类。Page 类有一个 type 属性，目前允许 2 种值，用于分为绝对型和流型两种方式提供页面布局方法。

每个 Page 类中将包含若干个 Panel 类。Panel 也是一种 Poi 语言的内置类，它是容纳 HTML DOM 元素的基本容器，所有 DOM 元素由 Panel 包含。同级的各 Panel 占据一定的空间，不能重叠，但是 Panel 类可以包含其它 Panel 类，以提供树形的页面布局支持。

页面布局的所有操作，均可通过对内置类的属性取值/赋值，以及内置方法调用完成。

8.1.1 绝对型 Page 布局

若 Page.Type=Abs，则 Poi 语言将按照绝对定位方式提供页面布局支持。

此时，Page 将提供 2 个属性 CntRow 和 CntCol 供使用者指定，页面被划分为 CntRow 行 CntCol 列的网格。Page 类提供一个类型为 Map 的属性，包含 Page 中的各个 Panel 及其占据的位置。每个 Panel 可以占据若干个网格构成的矩形，在 Panel 被添加到 Map 时，由 Page 类自动检查与其它 Panel 定位的冲突，并告知使用者。

8.1.2 流型 Page 布局

若 Page.Type=Flow，则 Poi 语言将按照流定位方式提供页面布局支持。

此时，Page 类用于网格划分的属性无效，Map 包含的各 Panel 将按照其被加入 Map 的时间次序被依次添加到页面中，每个 Panel 均占据页面的整个宽度，并根据 Panel 内部的内容和设定来决定高度。

8.1.3 内部 Panel 布局

Panel 同样包含一个 Type 属性，支持 Abs， Flow 两个值，分别表示使用绝对布局和流布局方式。只有父容器也是绝对布局时，绝对布局方式才被允许。

Panel 包含的 Map 类型的属性可以容纳子 Panel 类、HTML DOM 元素类或者另一种 Poi 语言的内置类：Group。

Group 是一个弱化的 Panel，只有 CntRow 和 Map 两种属性，支持按列划分容器、按行使用流布局。Group 的 Map 中包含若干流布局的 Panel，每个 Panel 占据一部分连续的列。

Panel 的绝对布局方式与 Page 相同，Map 中的每个元素占据一部分矩形网格。

Panel 的流布局方式除了支持与 Page 类似的布局方法外，还可以使用 Group 类。每个 Group 类中的 Panel 全体按 Group 指定的比例占据当前 Panel 的完整宽度。

8.2 实现 HTML 标签

每一个 HTML 标签对应 Poi 语言中的一个内置类，这些类继承自一个名为 HtmlElement 的 Poi 基类。同一个 HTML 标签的不同 type 在 Poi 语言中也使用继承实现。这些内置类只包含 HTML 标签中功能性的部分，所有布局和样式均由上述两节实现。除了大体上复制 HTML 标签包含的属性外，每个 HTML 标签对应的内置类还会提供一些方法，支持对这些 HTML 标签进行基本、常见的操作。

总而言之，HTML DOM 元素的创建可以由构建一个类的实例，并为其属性赋值来完成，通过该类的实例可以直接获得 DOM 中对应元素的对象引用，对其属性进行取值和修改。直接调用类的方法可以对 DOM 元素动态地进行操作。

8.3 实现 CSS 样式表

Poi 语言在设计时选取常用的 CSS 样式进行了分类，实现为若干个类。每个类提供自动转化为 Map 的函数，以及一些使书写更加简便的赋值方法。Poi 语言还允许自建键、值均为 string 的 Map 直接书写 CSS 样式。

CSS class 和 style 在 Poi 语言中均得到了支持。Poi 语言为 HtmlElement 基类包含一个类型为 Array[Map]的属性和一个类型为 Map 的属性。允许向前者中添加 Map，每个 Map 对应一个 CSS class。Array[Map]中的 CSS class 将被关联到 HTML 标签中。允许向后者中添加 string 键值对，Poi 语言将其编译该标签实例的 style 格式。显然，Map 中的样式优先于 Array[Map]。

9. 动态页面交互库

9.1 页面动态更新

在 Poi 代码中对 HTML 标签所对应类的修改、对 Page 类与 Panel 类布局属性的修改、对 CSS 样式 Map 的修改均会被编译为操作 DOM 元素的 JavaScript 代码反映在页面中。

9.2 用户动作检测

Poi 语言为 HtmlElement 基类包含 Hover 和 Click 两个 Event，用于注册当元素被经过或点击时执行的事件，它们同样被编译为 JavaScript 代码动态向 DOM 元素注册事件。

10. 动态服务器交互库

10.1 数据传输

在 Poi 语言编译生成的前端代码中，使用 JSON 数据格式与服务器进行交互。

由于 Poi 语言容器类型中的 Map 是一个字符串到任意类型的检索表，因此它可以原生支持对于 JSON 数据的操作。除此之外也可指定直接使用二进制流等其它基本传输方式。

10.2 HTTP 请求与 AJAX 实现

Poi 语言内置一个 HttpResponse 类，提供处理 HTTP 响应的支持。HttpResponse 类包含状态码等 HEAD 信息、返回内容等基本 HTTP 属性。

Poi 语言内置一个 HttpRequest 类，提供发送 HTTP 请求的支持。HttpRequest 类包含请求类型、地址、提交内容、是否跨域等基本配置属性，用于创建、修改、清除、发送等操作的内置方法。

为了支持 AJAX，HttpRequest 类还将包含 OnSucc，OnFail，OnEach 三个 Event，分别在该请求得到正确响应、错误响应、每次响应时调用 Event 中的函数。通过向这三个 Event 注册映射类型，即可实现回调函数。这三个 Event 包含的内容还有一个完整的 HttpResponse 类等资源供回调函数访问。

AJAX 的 HTTP Request 将被编译生成为 jQuery 库中一个强大的底层函数$.ajax()。