Poi-lang 项目结构

项目实现的主要部分位于 poi-lang\PoiCSharpAnalyzer\Implement 文件夹下：

语法分析：

LL(k)文法，poi.grammar 文件；

语义分析：

深度优先搜索语法树，创建 PoiObject 类，在每个节点上附带一个 List，存储综合属性与继承属性，PoiBasicAnalyzer.cs 文件；

类型检查：

在变量定义和使用时根据 PoiType 类检查，PoiAnalyzerScope.cs 文件；

网页设计：

PoiHtmlLayout.cs 文件。

在如下的语言手册中，将针对每一部分讲述语法与对应的实现方案。

Poi 语言手册

李煜东 李源昊 田菁曳

目录

1 Poi 语言概况

1.1 语言概述

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。

1.2 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 Analyzer Tools 进行，详细使用方法请参见《Poi Language Tools Manual》。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 / false

2.4 类型

2.4.1 基本类型

2.4.1.1 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

2.4.1.2 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

2.4.1.3 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。

2.4.2.1 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；

2.4.2.2 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；

2.4.2.3 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；

2.4.2.4Event

存储若干相同参数的函数的一个事件，当发生事件是可以以一个相同参数调用并执行其中的全部函数。

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> [= <init_val>]

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

init_val: 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后续章节中进行描述；

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（最高优先级）	(), ., f(), a[], ++, --(后缀)
2	+, -, !, ~, ++, --(前缀), T()
3	*, /, %
4	+, -
5	<<, >>
6	<, >, <=, >=
7	==, !=
8	&

9	^
10	|
11	&&
12	||
13	?:
14（最低优先级）	=, +=, -=, *=, %=, <<=, >>=, &=, |=, ^=

3.2.1 函数调用，数组取值，自增与自减

(): 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 struct 等数据结构中获取对应变量；（有关于 struct 将在网络部分详细说明）

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

+, -: 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）；

3.2.3 基本算数运算

*, /, %: 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

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

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
    init (<init_stmt_block>)
    while (<start_condition>)
    step (<step_stmt_block>)
    <loop_stmt_block>
    until (<end_condition>);
```

其中：

init 语句：初始化语句：整个循环开始前执行；语句间由分号隔开；

while 语句：循环条件：每次执行循环体前判断；

step 语句：步进语句：每次在循环体后执行；语句间由分号隔开；

<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_size1>, <array_size2>, …] <array_name>

其中：

<array_type>: 数组中的元素类型；

<array_size1>, <array_size2>, … : 多维数组每一维的大小

<array_name>: 数组名称

Poi 语言中 array 的定义支持任意多维的存储类型相同的元素的数组的定义，只需在定义时指明各维度的数组长度，例如：

array[int, 4, 5] test;

这个定义就等价于 C 语言中的 int[4][5] test;

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

一个事件(event)是一组有相同参数类型函数的集合。用户可以向一个 event 中添加函数，并在一定的时机以指定的参数调用这些函数。

定义:

```
event[<para1>, <para2>, …] <event_name>
```

添加函数:

```
<event_name> += <function_name> | <anonymous_function>
```

其中：

function_name 代表形如 function name = @[]->… 定义的函数，通过函数名进行添加与删除;

anonymous_function 是形如 @[]->… 的匿名函数，直接向 event 添加函数体，但不能删除;

删除:

```
<event_name> -= <function_name>
```

特别地，匿名函数不能被删除;

执行:

```
poi~ <event_name>(<para1>, <para2>, …)
```

以<para1>, <para2>, … 作为参数依次执行全部的函数（执行顺序与添加顺序相同）;

6 网页结构 Struct

结构(struct)是 poi 语言网页设计的基本单位。

Struct 共有 15 种基类，前 4 种用于静态页面布局，每种对应若干个 HTML 标签及其内容；后 11 种通过灵活的分类方式，涵盖了所有基本的 HTML 标签。

声明方式:

```
struct <name> as <base_name> {
    <property1> <value1>
    <property2> <value2>
    ...
 }
```

Struct 定义主要基于“单继承”。15 种基类属于内置定义，可直接作为 base_name 被编译器识别。随后编写者自定义的 struct 可以作为新的基类再次被继承。继承者自动获得被继承者的所有属性，若重复定义，则视为 override，使用继承者的新定义覆盖原定义。

6.1 Page, Grid, Panel, Group 基类

对应静态网页布局的 struct 有 Page,Grid,Panel,Group 四种。

每个 Page 对应一个.html 文件，包含完整的<html><head><body>标签。

Grid 提供绝对布局支持，可指定 2 个属性 CntRow 和 CntCol，页面被划分为 CntRow 行 CntCol 列的网格。在 Grid 中添加的每个子 struct 可以占据若干个网格构成的矩形。Grid 中的子 struct 的位置相对于该网格都是 position:absolute 的。

Group 提供流型分列布局支持，可指定属性 CntCol，页面被划分为等宽的 CntCol 列，每列在网页流中根据其内容多少具有自动高度，子 struct 可占据若干连续的列。

Panel 提供流行布局支持，其子 struct 按照添加顺序从上到下依次排布在网页上，每个子 struct 占据页面全宽。

对于按块展示型布局的网页设计，可用 Page→Grid→Panel 的包含关系轻松实现。

对于一般的左右分栏、上下滚动的内容型网页设计，可用 Page→Panel→Group→Panel 的包含关系轻松实现。

6.2 富文本：Text 基类

文本是大部分网页中必不可少的内容。每个 Text 结构具有两个基本属性：文本(value)、格式(format)。Text 被翻译为带格式的 html 。

value 的值作为的核心内容，通过 encode 属性，还可以让 poi 语言自动对文本进行 HTML 编码转义。

format 用于对文本进行格式化，全格式形如"/biu-+_^><@..."，每一位分别对应网页注释和 a, b, i, u, del, ins, sub, sup, big, small 这些格式化标签。

例如 struct example as Text {

format “b<@http://www.google.com.sg”

text “Google”

}; 就会被编译为

<small>Google</small>

6.3 表格→二维数组：Table 基类

把数据映射到网页表格中，是一个很常见却很麻烦的事情。在 HTML 中，原本结构化的数据却必须要用<tr><th><td>和 colspan/rowspan 包裹。

在 Poi 语言中，可以指定 Table 结构的长宽，并把子 struct 添加到 Table 的若干个连续的位置构成的矩形中，像二维数组一样拿到 Table 每个格子的对象，非常方便。

编译器的实现方式为向表格中每个标签追加 title 属性，标识其行、列，通过 jQuery 选择器选定对应 name 的表格中，相应位置对应的 title 标识的对象。

6.4 Input,Textarea 基类

与富文本 Text 类似，Input 与 Textarea 包含文本内容与一个权限格式化串。通过 permission 属性(形如”rxqac”)可以简洁地指定 readonly,disable 等权限。

6.5 Part,Single,Form,Image,Script,Button 基类

类似地，Form 实现表单、Image 实现图片、Script 实现内嵌代码、Button 实现按钮，Part 实现其余无特殊规定的双标签（通过 type 指定类型）、Single 则是单标签。

例如 struct h2 as Part{ type “h2” };定义<h2>。

此后，struct example as h2 { text “example”}就定义了<h2>example</h2>。

7 网页设计

7.1 静态网页布局

与通过“HTML 编写”和“JS 动态创建”一样，poi 语言同样支持两种编写方法。有一部分语法为静态操作，被编译为 HTML 文件，直接由浏览器渲染加载。另一部分语法编译为 JS 代码，通过浏览器实时执行，创建、修改和操作网页上的标签对象。

Add 与 generate 就是两个被编译为 HTML 文件的静态操作。

7.1.1 静态添加 Add

每个 Struct 具有 add 成员函数。

语法：struct1.add([struct2, args…]);

每个 struct 包含一个子结构 List，存储它包含的所有子 struct。

7.1.2 静态生成 Generate

每个 Struct 具有 generate 成员函数，但是只有 Page 类的 generate 可以被用户显式调用。

调用一个 struct 的 generate 后，将会把以该 struct 为根的整棵 struct 树中的结构编译为 HTML，并存储在内部属性 htmlcode 中。当 Page 类执行 generate 时，整棵树的 htmlcode 被写入以 Page 的 name 命名的 html 文件中。

7.2 动态网页操作

7.2.1 控制流化网页元素：struct 在 JS 中的对应

可以看出，无论是 javascript 还是 poi 语言，都是一种结构化、对象化的非标记式语言，其变量是动态的、具有特定作用域的。而 HTML 是标记式语言，其标签是嵌套的并且全局可见。

例如，在 poi 的 for 循环中，循环体内定义一个 struct。这个 struct 的作用域在循环体内，每次 for 循环时应该是不同的，但是如果直接翻译，它们在网页中将具有同样的名称，这显然是不符合设计初衷的。在 poi 中定义的 struct 如何通过 js 与 HTML 上的对象建立联系，是一个重要的问题。

解决方法是：把每个 struct（设其名称为 name）在 JS 中编译为一个整数变量（记为 cnt），在每次新定义 struct 时，变量的值 cnt++，并定义 var name = cnt; 向 HTML 中添加元素时，记录附加属性 title=cnt。

由于在不同作用域中的 struct 具有不同的整数变量值，它们在网页中也就具有不同的 title 属性。在某个作用域中，只需要获取当前 name 变量的值，就得到了该作用域下的 name 对应的 title。由于 struct 在 JS 中仅仅是一个整数变量，因此把网页元素对象作为函数参数、数组内容等都可以被实现了。

这样，标记式的网页结构就被转化成了控制流化的 poi 对象。

7.2.2 Global 与 Local 操作

Poi 语言中的所有动态网页操作分为两种类型。gfunc 为 Global 类型，a.gfunc 将对所有 name 属性为 a 的 DOM 元素调用 gfunc 函数。Lfunc 为 Local 类型，a.lfunc 将对上一步中提到的，title 属性为当前作用域下的 name 变量的值的 DOM 元素调用 gfunc 函数。

7.2.3 追加、清空、更新、显示与隐藏

向末尾追加子元素：gappend, lappend

清空子元素：gclear, lclear

更新元素内容：gtext, ltext, gurl, lurl, gperm, lperm

显示/隐藏元素：gshow, lshow, ghide, lhide

因此，如果通过 poi 语言创建一个 table，使用两重循环遍历它，在循环内创建一个 name 为 a 的 struct，通过 table.gappend([a, i, j])可将 a 追加到表格的第 i 行第 j 列中。

如果我们顺便把 a 放入二维数组 arr[i,j]=a，那么需要修改表格时，调用 a.gtext 将修改 arr 中的所有元素 a 的文本（因为它们具有相同的 struct name）。而循环数组 arr，对每个 arr[i,j]调用 ltext，则只有指定的那个 DOM 元素会被更新（因为它们的 title 属性各不相同，都是循环(i,j)作用域下变量 name 的值）。

7.3 CSS 样式表

7.3.1 样式表操作

style <style_address> - 通过设置 page 属性链接外部样式表

<element_name>.css(<map_name>) - 通过 map 设置元素的 css 属性

<element_name>.cssadd([<name>, <value>]) - 向元素添加 css 属性

<element_name>.cssdel(<name>) - 从元素删除 css 属性

<element_name>.classadd(<class_name1 class_name2 …>) - 向元素添加一个或多个类

<element_name>.classdel(<class_nam1e class_name2 …>) - 从元素删除一个或多个类

<element_name>.classtoggle(<class_name class_name2 …>) - 对元素进行添加/删除一个或多个类的切换操作

7.3.2 外部样式表

style <style_sheet_address>

style_sheet_address 为外部样式表的路径

实例：

```
struct index as Page {
      title “index"
      style “style.css”
};
```

将 Page index 链接到外部样式表 style.css

7.3.3 通过 map 设置 CSS

<element_name>.css(<map_name>)

map_name 存放了所有形如<name, value>的样式

实例：

```
map tc;
tc["width"] = 800;
tc["height"] = tc["width"];
tc["font-size"] = 50;
table.css(tc);
```

将元素 table 的宽高都设为 800，字体大小设为 50。

7.3.4 添加 CSS 属性

<element_name>.cssadd([<name>, <value>])

name 规定 css 属性的名称；

value 规定 css 属性的值。

实例：

```
table.cssadd(["background-color", “red"]);
```

将元素 table 的背景设为红色。

7.3.5 删除 CSS 属性

<element_name>.cssdel(<name>)

name 规定了所要删除的 css 属性的名称。

实例：

```
p.cssdel(“color”);
```

删除元素 p 的 color 属性。

7.3.6 操作 CSS 属性类

```
.important
{
      font-weight:bold;
      font-size:60px;
}
.blue
{
      color:blue;
}
```

上述样式表将被应用于下述三个样例：

<element_name>.classadd(<class_name1 class_name2 …>)

class_name 规定了所要添加的 class 名称

```
text.classadd(“important blue”);
```

向元素 text 添加 important 和 blue 属性

<element_name>.classdel(<class_name1 class_name2 …>)

class_name 规定了所要删除的 class 名称

text.classdel(“important”);

从元素 text 删除 important 属性

<element_name>.classtoggle(<class_name1 class_name2 …>)

class_name 规定了所要切换的 class 名称

text.classtoggle(“important”);

切换元素 text 的 important 属性

8 用户交互事件

8.1 动作列表

<element_name>.bind([<action>, <event_name>, <event_parameter1>, <event_parameter2>, …]) - 向动作绑定事件

<element_name>.erase(<action>) - 从动作移除已添加的事件

<element_name>.click([<event_name>, <event_parameter1>, <event_parameter2>, …]) - 将事件绑定到 click 动作

<element_name>.dblclick([<event_name>, <event_parameter1>, <event_parameter2>, …]) - 将事件绑定到 double click 动作

<element_name>.focus([<event_name>, <event_parameter1>, <event_parameter2>, …]) - 将事件绑定到 focus 动作

<element_name>.mouseover([<event_name>, <event_parameter1>, <event_parameter2>, …]) - 将事件绑定到 mouseover 动作

8.2 绑定 bind

<element_name>.bind([<action>, <event_name>, <event_parameter1>, <event_parameter2>, …])

action 规定添加到元素的动作

event_name 规定了绑定到动作的事件

event_parameter1, event_parameter2, … 规定了事件所需要的参数

实例：

t.bind(["click", e, i, j]);

将事件 e 绑定到元素 t 的动作 click，事件 e 的参数为[i, j]。

8.3 清除 erase

<element_name>.erase(<action>)

action 规定添加到元素的动作

实例：

t.erase(“click");

删除元素 t 的动作 click 所绑定的事件。

8.4 单击事件 click

<element_name>.click([<event_name>, <event_parameter1>, <event_parameter2>, …])

event_name 规定了绑定到动作的事件

event_parameter1, event_parameter2, … 规定了事件所需要的参数

实例：

t.click([e, i, j]);

元素 t 发生动作 click 时触发事件 e，事件 e 的参数为[i, j]。

注意：t.click([e, i, j])和 t.bind(["click", e, i, j])的区别。前者在事件发生时传入 i, j 的值，而后者则在绑定事件时传入 i, j 的值，因此在循环变量为 i, j 的循环语句中，必须使用后者。

9 服务器交互 AJAX

9.1 语法

poi~ ajax_request([method, url, data, cors, resp_succ, resp_err, extra_config]);

含义为：

以 method 方法（Get/Post/Put/Delete 等）、向地址 url、

发送 JSON 数据 data（在 poi 语言中为 map 或 array）、是/否为跨域请求、

成功响应时的 poi 回调 Event、失败响应式的 poi 回调 Event、

其它不常用的设定表（一个 poi map）。

9.2 实现

基于 Event 语法与在 jQuery 库中的$.ajax()函数实现：

```
function __poi_ajax_request(method, url, data, cors, resp_succ, resp_err, extra_config) {
    var config = {
        method: method,
        url: url,
        data: data,
        success: function (data) { resp_succ.execute(data); },
        error: function (xhr, err) { resp_err.execute.call(xhr, err); }
    };
    for (var key in extra_config) {
        config[key] = extra_config[key];
    }
    if (cors) config.xhrFields.withCredentials = true;
    $.ajax(config);
}
var __poi_ajax = new Event();
__poi_ajax.add('', __poi_ajax_request);
```