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 4](#_Toc439974359)

[1.1 语言概述 4](#_Toc439974360)

[1.2 语言特点 4](#_Toc439974361)

[2 基础特性 4](#_Toc439974362)

[2.1 语句简介 4](#_Toc439974363)

[2.2 编译与执行 4](#_Toc439974364)

[2.3 基本常量 4](#_Toc439974365)

[2.4 类型 4](#_Toc439974366)

[2.4.1 基本类型 4](#_Toc439974367)

[2.4.2 容器类型 5](#_Toc439974368)

[3 定义、赋值、运算 5](#_Toc439974369)

[3.1 定义与赋值 5](#_Toc439974370)

[3.1.1 变量定义 5](#_Toc439974371)

[3.1.2 映射类型(function) 6](#_Toc439974372)

[3.1.3 赋值 6](#_Toc439974373)

[3.2 基本运算 6](#_Toc439974374)

[3.2.1 函数调用，数组取值，自增与自减 7](#_Toc439974375)

[3.2.2 单目运算符 7](#_Toc439974376)

[3.2.3 基本算数运算 7](#_Toc439974377)

[3.2.4 比较运算 7](#_Toc439974378)

[3.2.5 位运算 7](#_Toc439974379)

[3.2.6 逻辑运算 8](#_Toc439974380)

[3.2.7 三目运算符 8](#_Toc439974381)

[3.2.8 赋值运算 8](#_Toc439974382)

[4 控制结构语句 8](#_Toc439974383)

[4.1 分支语句 8](#_Toc439974384)

[4.2 循环语句 8](#_Toc439974385)

[4.3 返回语句 9](#_Toc439974386)

[5 容器类型 9](#_Toc439974387)

[5.1 序对 pair 9](#_Toc439974388)

[5.2 数组 array 9](#_Toc439974389)

[5.3 检索表 map 9](#_Toc439974390)

[5.4 事件 event 10](#_Toc439974391)

[6 网页结构 Struct 10](#_Toc439974392)

[6.1 Page, Grid, Panel, Group基类 10](#_Toc439974393)

[6.2 富文本：Text基类 11](#_Toc439974394)

[6.3 表格→二维数组：Table基类 11](#_Toc439974395)

[6.4 Input,Textarea基类 11](#_Toc439974396)

[6.5 Part,Single,Form,Image,Script,Button基类 11](#_Toc439974397)

[7 网页设计 12](#_Toc439974398)

[7.1 静态网页布局 12](#_Toc439974399)

[7.1.1 静态添加 Add 12](#_Toc439974400)

[7.1.2 静态生成 Generate 12](#_Toc439974401)

[7.2 动态网页操作 12](#_Toc439974402)

[7.2.1 控制流化网页元素：struct在JS中的对应 12](#_Toc439974403)

[7.2.2 Global与Local操作 13](#_Toc439974404)

[7.2.3 追加、清空、更新、显示与隐藏 13](#_Toc439974405)

[7.3 CSS样式表 13](#_Toc439974406)

[7.3.1 样式表操作 13](#_Toc439974407)

[7.3.2 外部样式表 13](#_Toc439974408)

[7.3.3 通过map设置CSS 14](#_Toc439974409)

[7.3.4 添加CSS属性 14](#_Toc439974410)

[7.3.5 删除CSS属性 14](#_Toc439974411)

[7.3.6 操作CSS属性类 14](#_Toc439974412)

[8 用户交互事件 15](#_Toc439974413)

[8.1 动作列表 15](#_Toc439974414)

[8.2 绑定 bind 15](#_Toc439974415)

[8.3 清除 erase 15](#_Toc439974416)

[8.4 单击事件 click 16](#_Toc439974417)

[9 服务器交互 AJAX 16](#_Toc439974418)

[9.1 语法 16](#_Toc439974419)

[9.2 实现 16](#_Toc439974420)

# Poi语言概况

## 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。

## 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

# 基础特性

## 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；

## 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；

## 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

## 类型

### 基本类型

#### 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 |

#### 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

#### 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；

### 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。

#### 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；

#### 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；

#### 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；

#### Event

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。

# 定义、赋值、运算

## 定义与赋值

### 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；

### 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；

### 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；

## 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） | =, +=, -=, \*=, %=, <<=, >>=, &=, |=, ^= |

### 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）；

### 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）；

### 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）；

### 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）；

### 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(|)

### 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；

### 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

### 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）

# 控制结构语句

## 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；

## 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；

## 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；

# 容器类型

## 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；

## 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;

## 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

## 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）;

# 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。

## 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。

## 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 <span>。

value的值作为<span>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

<a href=”http://www.google.com.sg”><b><small>Google</small></b></a>

## 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。

## 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。

## 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>。

# 网页设计

## 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。

### 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。

### 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。

## 动态网页操作

### 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。

### 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。

### 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）。

## CSS样式表

### 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

### 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

### 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。

### 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。

### 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。

### 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

# 用户交互事件

## 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

## 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]。

## 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。

## 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。

# 服务器交互 AJAX

## 语法

poi~ ajax\_request([method, url, data, cors, resp\_succ, resp\_err, extra\_config]);

含义为：

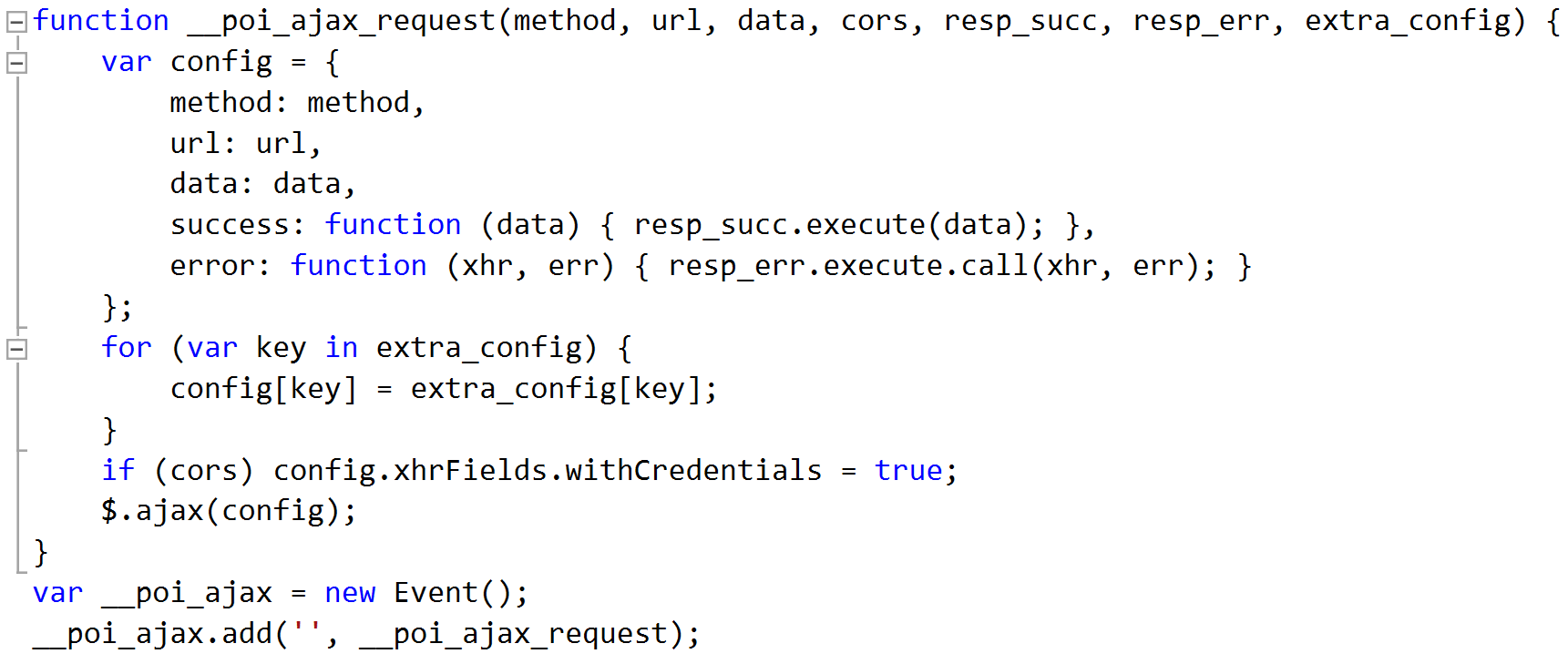
以method方法（Get/Post/Put/Delete等）、向地址url、

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）。

## 实现

 基于Event语法与在jQuery库中的$.ajax()函数实现：