预处理

PlantUML包含了一些辅助性的预处理功能, 并且适用于*所有*的图.

这些功能与[C language preprocessor](https://en.wikipedia.org/wiki/C_preprocessor)很相似, 除了标记由#替换为!.

迁移说明

目前的预处理是由legacy preprocessor.升级而来.

虽然一些历史遗留功能仍被目前的预处理支持, 但是你不应该继续使用 (他们不久将会被移除).

- 你不应该再使用 `!define` 和 `!definelong`. 使用 `!function` 和定义变量替换他们. `!define`替换为 返回函数 而 `!definelong` 应该替换为 void function.
- `!include` 现在允许多包含：不应该再使用 `!include_many`
- `!include` 现在可以授受URL, 所以不再需要 `!includeurl`
- 一些特性 (比如 `%date%`) 替换为内建函数 (例如 `%date()`)
- 当不带参数调用历史遗留`!definelong` 宏的时候, 你必须使用括号. 必须使用 `my_own_definelong()` 因为 `my_own_definelong`不带括号的形式不被新的预处理语法解析.

如果你有什么疑问请联系我们.

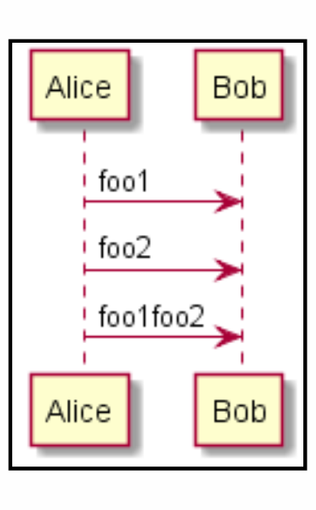
定义变量

虽然这还是必须的, 我们强烈建议变量名以 `$`开头. 有两类数据类型:

- 整□型
- 字符串 - *必须*被单引号或双引号包围.

在函数外创建的变量作用域是**global**, 你可以在任何地方访问他们 (包括函数).当定义变量的时候你可以使用 `global`强调这一点.

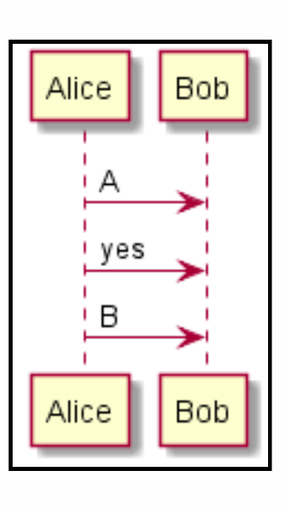
```
@startuml
!$ab = "foo1"
!$cd = "foo2"
!$ef = $ab + $cd

Alice -> Bob : $ab
Alice -> Bob : $cd
Alice -> Bob : $ef
@enduml
```



条件

- 可以在条件里使用表达式.
- 支持*else*语法

```
@startuml
!$a = 10
!$ijk = "foo"
Alice -> Bob : A
!if ($ijk == "foo") && ($a+10>=4)
Alice -> Bob : yes
!else
Alice -> Bob : This should not appear
!endif
Alice -> Bob : B
@enduml
```

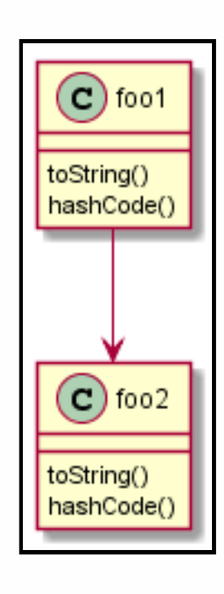


空函数

- 函数名 *必须* 以`$`开头
- 参数名 *必须* 以 `$`开头
- 空函数可以调用其他空函数

例:

```
@startuml
!function msg($source, $destination)
$source --> $destination
!endfunction

!function init_class($name)
class $name {
$addCommonMethod()
}
!endfunction


!function $addCommonMethod()
  toString()
  hashCode()
!endfunction


init_class("foo1")
init_class("foo2")
msg("foo1", "foo2")
@enduml
```



函数里定义的变量作用域为**local**. 意味着随函数一同销毁.

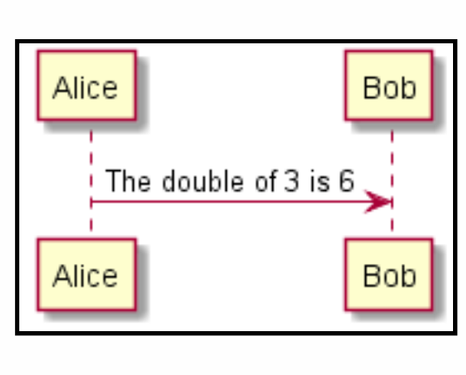
返回函数

返回函数不输出任何东西. 它只是定义了一个你可以调用的函数:

- 直接在变量和图中文本中使用
- 被其他返回函数调用
- 被其他空函数调用

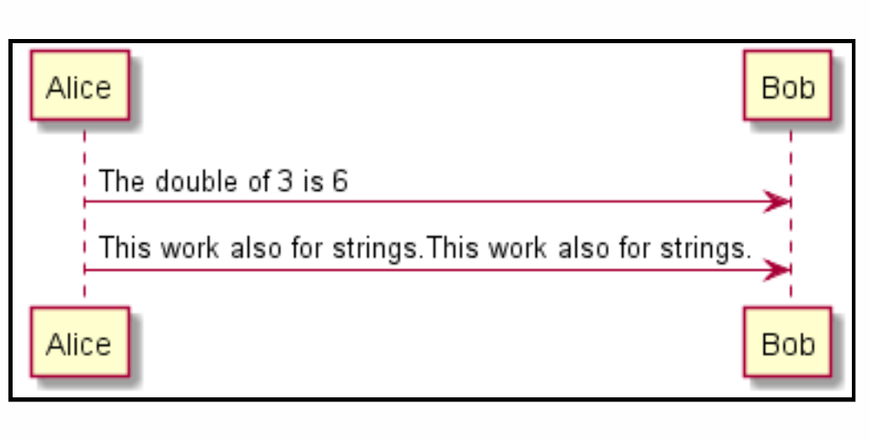
- 函数名 *应该* 以一个`$`开头
- 参数名 *应该* 以一个 `$`开关

```
@startuml
!function $double($a)
!return $a + $a
!endfunction

Alice -> Bob : The double of 3 is $double(3)
@enduml
```



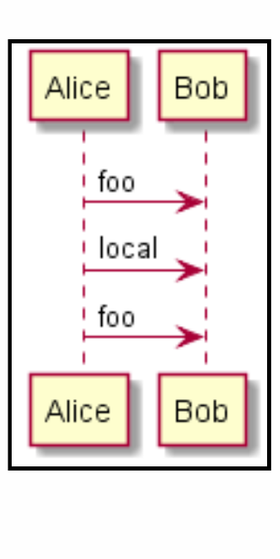
可以简化函数定义为一行:

```
@startuml
!function $double($a) return $a + $a

Alice -> Bob : The double of 3 is $double(3)
Alice -> Bob : $double("This work also for strings.")
@enduml
```



像空函数一样, 变量默认为'local'本地变量 (随函数退出销毁). 并且, 你可以在函数中访问'global'全局变量. 并且, 如何一个全局变量已存在,你仍可以使用 `local` 关键字创建一个同名的本地变量.

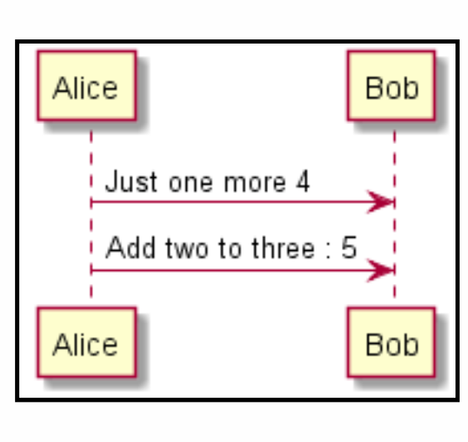
```
@startuml
!function $dummy()
!local $ijk = "local"
Alice -> Bob : $ijk
!endfunction

!global $ijk = "foo"

Alice -> Bob : $ijk
$dummy()
Alice -> Bob : $ijk
@enduml
```



参数默认值

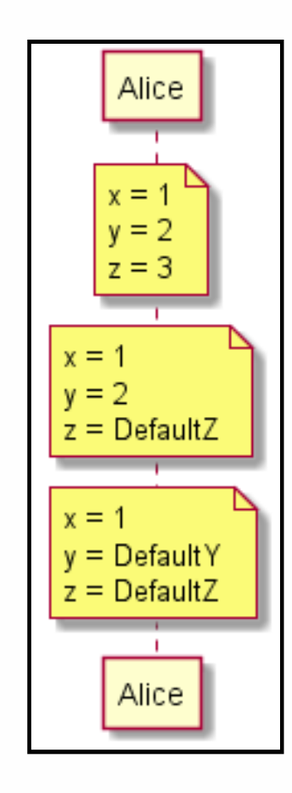
在返回和空函数中, 你可以定义参数默认值.

```
@startuml
!function $inc($value, $step=1)
!return $value + $step
!endfunction

Alice -> Bob : Just one more $inc(3)
Alice -> Bob : Add two to three : $inc(3, 2)
@enduml
```



只有在尾端的参数列表才可以定义默认值.

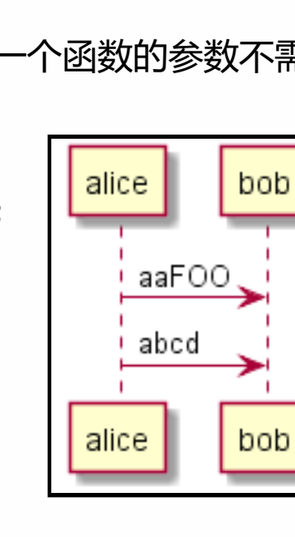
```
@startuml
!function defaulttest($x, $y="DefaultY", $z="DefaultZ")
note over Alice
  x = $x
  y = $y
  z = $z
end note
!endfunction

defaulttest(1, 2, 3)
defaulttest(1, 2)
defaulttest(1)
@enduml
```



非引号函数

默认情况下, 调用一个函数需要引号. 可以使用 `unquoted` 关键字指明一个函数的参数不需要使用引号.

```
@startuml
!unquoted function id($text1, $text2="FOO") return $text1 + $text2

alice -> bob : id(aa)
alice -> bob : id(ab,cd)
@enduml
```

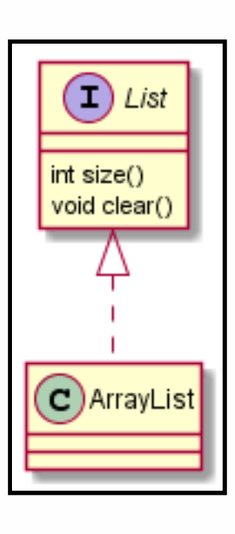


Including files or URL

Use the `!include` directive to include file in your diagram. Using URL, you can also include file from Internet/Intranet.

Imagine you have the very same class that appears in many diagrams. Instead of duplicating the description of this class, you can define a file that contains the description.

```
@startuml

interface List
List : int size()
List : void clear()
List <|.. ArrayList
@enduml
```



File List.iuml

```
interface List
List : int size()
List : void clear()
```

The file `List.iuml` can be included in many diagrams, and any modification in this file will change all diagrams that include it.

You can also put several `@startuml/@enduml` text block in an included file and then specify which block you want to include adding `!0` where `0` is the block number. The `!0` notation denotes the first diagram.

For example, if you use `!include foo.txt!1`, the second `@startuml/@enduml` block within `foo.txt` will be included.

You can also put an id to some `@startuml/@enduml` text block in an included file using `@startuml(id=MY_OWN_ID)` syntax and then include the block adding `!MY_OWN_ID` when including the file, so using something like `!include foo.txt!MY_OWN_ID`.

By default, a file can only be included once. You can use `!include_many` instead of `!include` if you want to include some file several times. Note that there is also a `!include_once` directive that raises an error if a file is included several times.

Including Subpart

You can also use `!startsub NAME` and `!endsub` to indicate sections of text to include from other files using `!includesub`. For example:

file1.puml:

```
@startuml

A -> A : stuff1
!startsub BASIC
B -> B : stuff2
!endsub
C -> C : stuff3
!startsub BASIC
D -> D : stuff4
!endsub
@enduml
```

file1.puml would be rendered exactly as if it were:

```
@startuml

A -> A : stuff1
B -> B : stuff2
C -> C : stuff3
D -> D : stuff4
@enduml
```

However, this would also allow you to have another file2.puml like this:

file2.puml

```
@startuml

title this contains only B and D
!includesub file1.puml!BASIC
@enduml
```

This file would be rendered exactly as if:

```
@startuml

title this contains only B and D
B -> B : stuff2
D -> D : stuff4
@enduml
```

Builtin functions

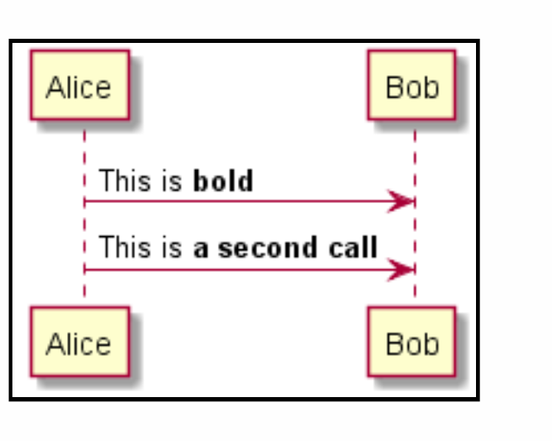
Some functions are defined by default. Their name starts by `%`

Name	Description	Example	Return
`%strlen`	Calculate the length of a String	`%strlen("foo")`	`3` in the example
`%substr`	Extract a substring. Takes 2 or 3 arguments	`%substr("abcdef", 3, 2)`	`"de"` in the example
`%strpos`	Search a substring in a string	`%strpos("abcdef", "ef")`	4 (position of `ef`)
`%intval`	Convert a String to Int	`%intval("42")`	42
`%file_exists`	Check if a file exists on the local filesystem	`%file_exists("c:/foo/dummy.txt")`	`true` if the file exists
`%function_exists`	Check if a function exists	`%function_exists("$some_function")`	`true` if the function has been defined
`%variable_exists`	Check if a variable exists	`%variable_exists("$my_variable")`	`true` if the variable has been defined exists
`%set_variable_value`	Set a global variable	`%set_variable_value("$my_variable", "some_value")`	An empty string
`%get_variable_value`	Retrieve some variable value	`%get_variable_value("$my_variable")`	the value of the variable
`%getenv`	Retrieve environment variable value	`%getenv("OS")`	The value of `OS` variable
`%dirpath`	Retrieve current dirpath	`%dirpath()`	Current path
`%filename`	Retrieve current filename	`%filename()`	Current filename
`%date`	Retrieve current date. You can provide an optional format for the date	`%date("yyyy.MM.dd at HH:mm")`	Current date
`%true`	Return always `true`	`%true()`	`true`
`%false`	Return always `false`	`%false()`	`false`
`%not`	Return the logical negation of an expression	`%not(2+2==4)`	`false` in that example
`%lower`	Return a lowercase string	`%lower("Hello")`	`hello` in that example
`%upper`	Return an uppercase string	`%upper("Hello")`	`HELLO` in that example

Logging

You can use `!log` to add some log output when generating the diagram. This has no impact at all on the diagram itself. However, those logs are printed in the command line's output stream. This could be useful for debug purpose.

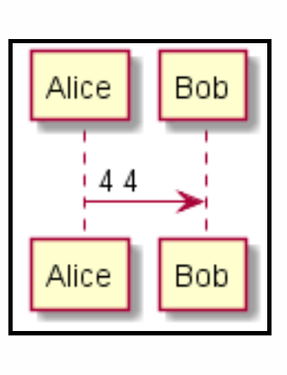
```
@startuml
!function bold($text)
!$result = "<b>"+ $text +"</b>"
!log Calling bold function with $text. The result is $result
!return $result
!endfunction

Alice -> Bob : This is bold("bold")
Alice -> Bob : This is bold("a second call")
@enduml
```



Memory dump

You can use `!memory_dump` to dump the full content of the memory when generating the diagram. An optional string can be put after `!memory_dump`. This has no impact at all on the diagram itself. This could be useful for debug purpose.

```
@startuml
!function $inc($string)
!$val = %intval($string)
!log value is $val
!dump_memory
!return $val+1
!endfunction

Alice -> Bob : 4 $inc("3")
!unused = "foo"
!dump_memory EOF
@enduml
```



Assertion

You can put assertion in your diagram.

```
@startuml
Alice -> Bob : Hello
!assert %strpos("abcdef", "cd")==3 : "This always fail"
@enduml
```

Building custom library

It's possible to package a set of included files into a single .zip or .jar archive. This single zip/jar can then be imported into your diagram using `!import` directive.

Once the library has been imported, you can `!include` file from this single zip/jar.

Example:

```
@startuml

!import /path/to/customLibrary.zip
' This just adds "customLibrary.zip" in the search path

!include myFolder/myFile.iuml
' Assuming that myFolder/myFile.iuml is located somewhere
' either inside "customLibrary.zip" or on the local filesystem

...
```

Search path

You can specify the java property `plantuml.include.path` in the command line.

For example:

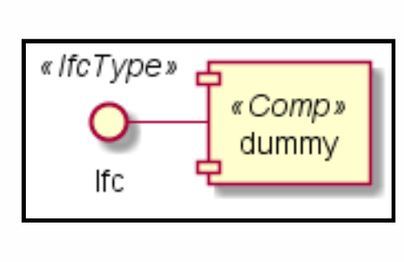
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

Note the this -D option has to put before the -jar option. -D options after the -jar option will be used to define constants within plantuml preprocessor.

Argument concatenation

It is possible to append text to a macro argument using the `##` syntax.

```
@startuml
!unquoted function COMP_TEXTGENCOMP(name)
[name] << Comp >>
interface Ifc << IfcType >> AS name##Ifc
name##Ifc - [name]
!endfunction
COMP_TEXTGENCOMP(dummy)
@enduml
```

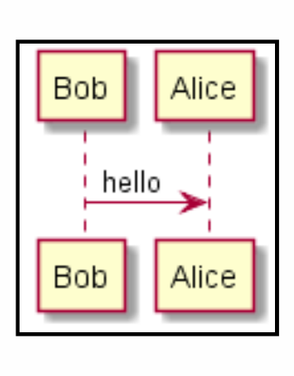


Dynamic function invocation

You can dynamically invoke a void function using the special `%invoke_void_func()` void function. This function takes as first argument the name of the actual void function to be called. The following argument are copied to the called function.

For example, you can have:

```
@startuml
!function $go()
 Bob -> Alice : hello
!endfunction

!$wrapper = "$go"

%invoke_void_func($wrapper)
@enduml
```



For return functions, you can use the corresponding special function `%call_user_func()` :

```
@startuml
!function bold($text)
!return "<b>"+ $text +"</b>"
!endfunction

Alice -> Bob : %call_user_func("bold", "Hello") there
@enduml
```