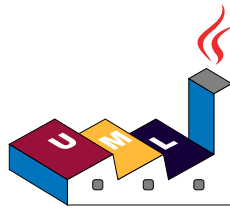


使用 PlantUML 绘制的 UML



语言参考指引

(2015 年 10 月 14 日星期三上午 8:53)

PlantUML 是一个开源项目，并支持快速绘制：

- 时序图
- 用例图
- 类图
- 活动图
- 组件图
- 状态图
- 对象图

以简单并带指引性语言定义各种视图。

1 时序图

1.1 简单示例

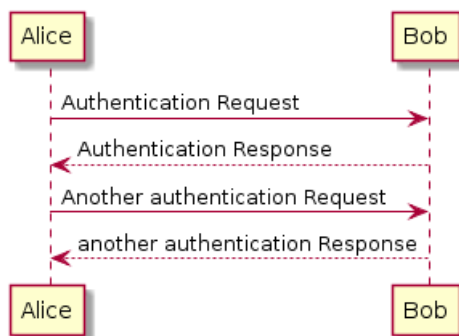
你可以使用 `->` 来绘制参与者之间的消息传递，而不必显式的声明参与者。

你也可以使用 `-->` 绘制一个虚线箭头表示异步消息。

另外，你也可以使用 `<-` 和 `<--`。这虽然不影响图形绘制，但可以增加可读性 Note that this is only true for sequence diagrams, rules are different for the other diagrams.

```
@startuml
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response
@enduml
```



1.2 注释

所有以单引号开头的行，都是注释

你也可以使用多行注释，多行注释以 `/'` 开头 `/'` 结尾。

1.3 声明参与者

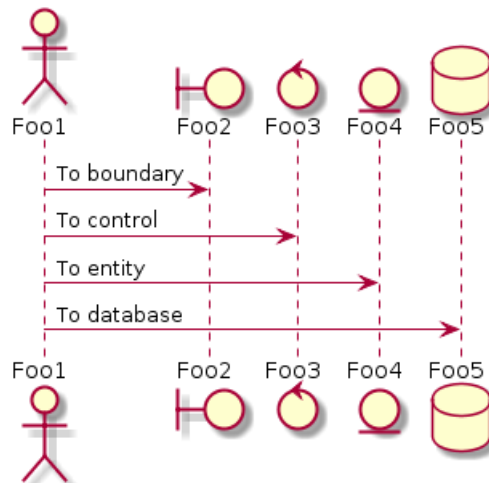
可以使用 `participant` 关键字来改变参与者的先后顺序。

你也可以使用其它关键字来声明参与者：

- actor
- boundary
- control
- entity
- database

```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
@enduml
```





使用 `as` 关键字重命名参与者

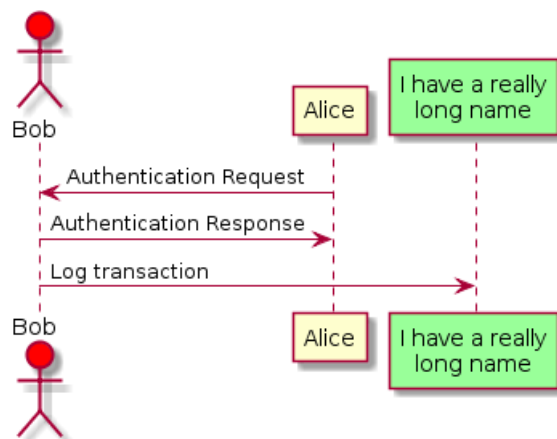
你可以使用 RGB 值或者颜色名修改 actor 或参与者的背景色。

```

@startuml
actor Bob #red
' The only difference between actor
'and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L #99FF99
/' You can also declare:
participant L as "I have a really\nlong name" #99FF99
'/
  
```

```

Alice->Bob: Authentication Request
Bob->Alice: Authentication Response
Bob->L: Log transaction
@enduml
  
```



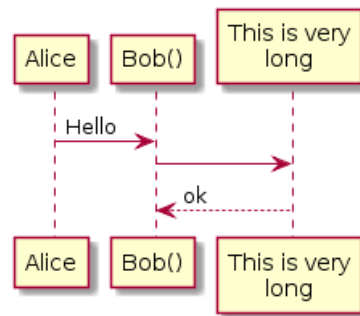
1.4 在参与者中使用非字符

你可以使用引号定义参与者。还可以用 `as` 关键字给参与者定义别名。

```

@startuml
Alice -> "Bob()" : Hello
"Bob()" -> "This is very\nlong" as Long
' You can also declare:
' "Bob()" -> Long as "This is very\nlong"
Long --> "Bob()" : ok
@enduml
  
```



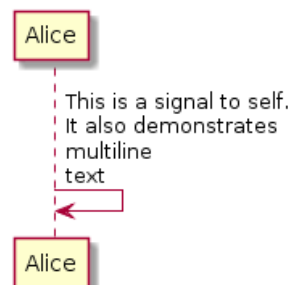


1.5 消息发给自己

参与者可以发消息给自己，
消息文字可以使用 \n 来换行。

```

@startuml
Alice->Alice: This is a signal to self.\nIt also demonstrates\nmultiline \ntext
@enduml
  
```



1.6 更改箭头的样式

修改箭头样式的方式有以下几种:

- 末尾加 x 表示一条丢失的消息
- 使用 \ 或 / 替代 < 或 > 来表示 have only the bottom or top part of the arrow
- 使用两个箭头标记 (如 >> 或 //) 表示空心箭头。
- 使用 -- 替代 - 以表示虚线箭头。
- 在箭头末尾加 “o”
- 双向箭头。

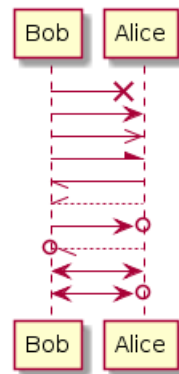
```

@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \- Alice
Bob //-- Alice

Bob ->o Alice
Bob o\-- Alice

Bob <-> Alice
Bob <->o Alice
@enduml
  
```

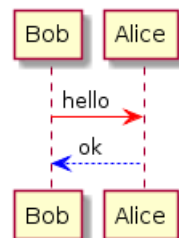




1.7 修改箭头颜色

你可以用以下语法修改箭头标记的颜色：

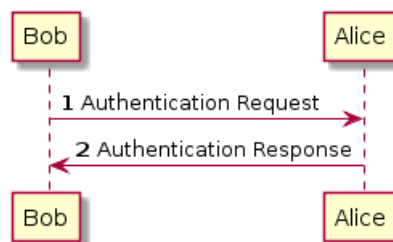
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.8 消息编号

autonumber 关键字用于自动的给消息加上编号。

```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response
@enduml
```



你还可以用 **autonumber 'start'** 的语法指定编号的初始值, 而用 **autonumber 'start' 'increment'** 可以同时指定编号的初始值和每次增加的值。

```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber 15
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

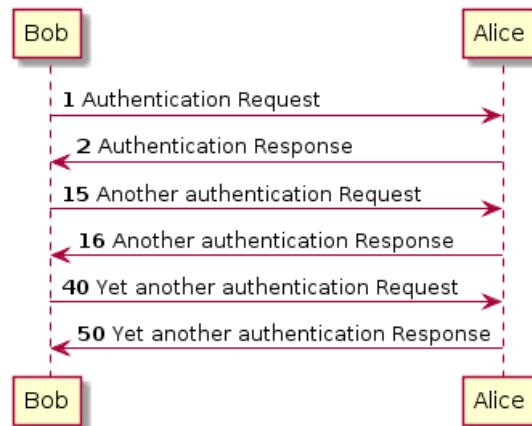
autonumber 40 10
Bob -> Alice : Yet another authentication Request
```



```

Bob <- Alice : Yet another authentication Response
@enduml

```



你可以在双引号内指定编号的格式。

格式是由 Java 的 `DecimalFormat` 类实现的 ('0' 表示数字, '#' 表示数字且默认为 0)。

你还可以使用 HTML 标签来制定格式。

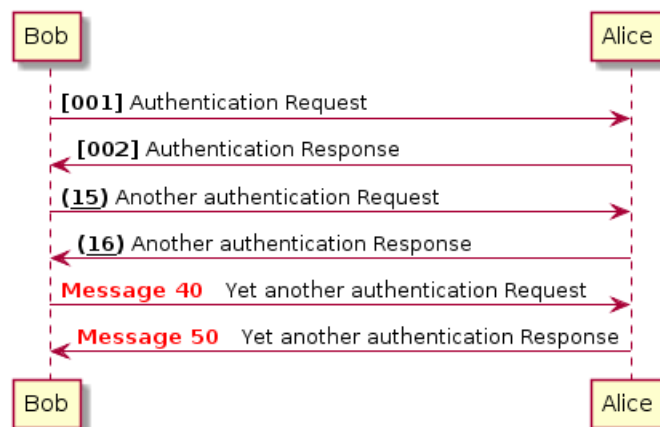
```

@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber 15 "<b>(<u>##</u>)"
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

autonumber 40 10 "<font color=red><b>Message 0 "
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response
@enduml

```



You can also use `autonumber stop` and `autonumber resume 'increment' 'format'` to respectively pause and resume automatic numbering.

```

@startuml
autonumber 10 10 "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber stop
Bob -> Alice : dummy

```



```

autonumber resume "<font color=red><b>Message 0  "
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

autonumber stop
Bob -> Alice : dummy

autonumber resume 1 "<font color=blue><b>Message 0  "
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response
@enduml

```

Simple communication example



1.9 标题 (Title)

`title` 关键字用于设置一个标题.

```

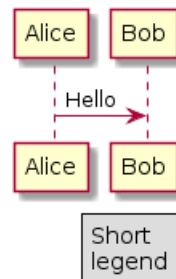
@startuml

title Simple communication example

Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

@enduml

```



1.10 给图表 (diagram) 添加备注

关键字 `legend` 和 `end legend` 用于添加备注。

可选项 `left`, `right` 和 `center` 可用于设置标注的对齐方式。

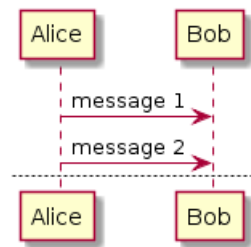
```

@startuml

Alice -> Bob : Hello
legend right
Short
legend
endlegend

@enduml

```



1.11 分割图表 (diagram)

关键字 **newpage** 用于把一个图表分割到多个图片中。

还可以在关键字 **newpage** 之后添加文字作为新图表的标题。
这便于在 *Word* 中打印多页的长图。

```
@startuml
```

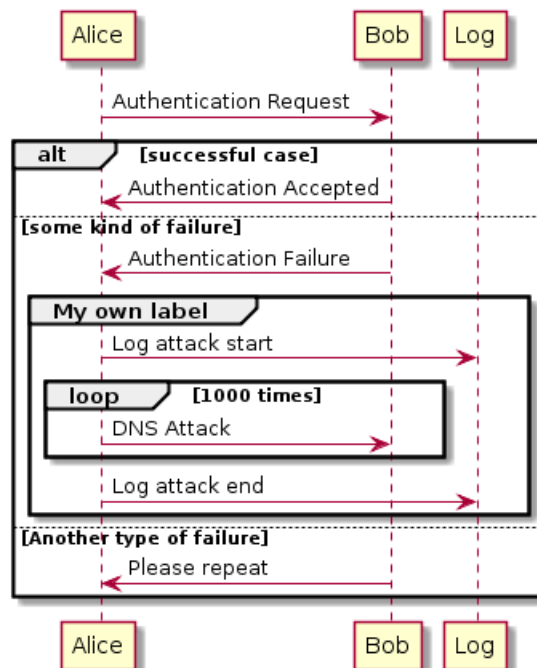
```
Alice -> Bob : message 1
Alice -> Bob : message 2
```

```
newpage
```

```
Alice -> Bob : message 3
Alice -> Bob : message 4
```

```
newpage A title for the\nlast page
```

```
Alice -> Bob : message 5
Alice -> Bob : message 6
@enduml
```



1.12 组合消息

我们可以通过以下关键词将组合消息:

- alt/else

- opt
- loop
- par
- break
- critical
- group, 后面紧跟着消息内容

可以在标头 (header) 添加需要显示的文字 (group 除外)。
 关键词 **end** 用来结束分组。
 注意, 分组可以嵌套使用。

```
@startuml
Alice -> Bob: Authentication Request

alt successful case

Bob -> Alice: Authentication Accepted

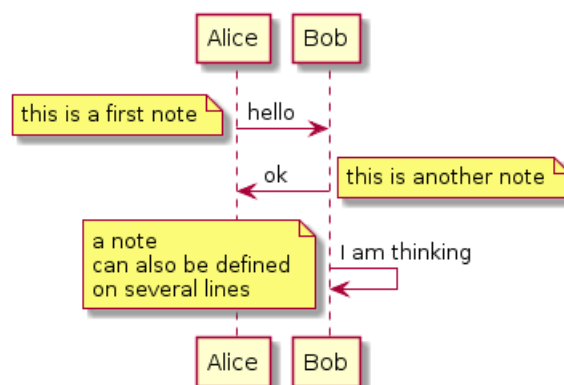
else some kind of failure

Bob -> Alice: Authentication Failure
group My own label
Alice -> Log : Log attack start
loop 1000 times
Alice -> Bob: DNS Attack
end
Alice -> Log : Log attack end
end

else Another type of failure

Bob -> Alice: Please repeat

end
@enduml
```



1.13 给消息添加注释

我们可以通过在消息后面添加 **note left** 或者 **note right** 关键词来给消息添加注释。
 你也可以通过使用 **end note** 来添加多行注释。

```
@startuml
Alice->>Bob : hello
note left: this is a first note

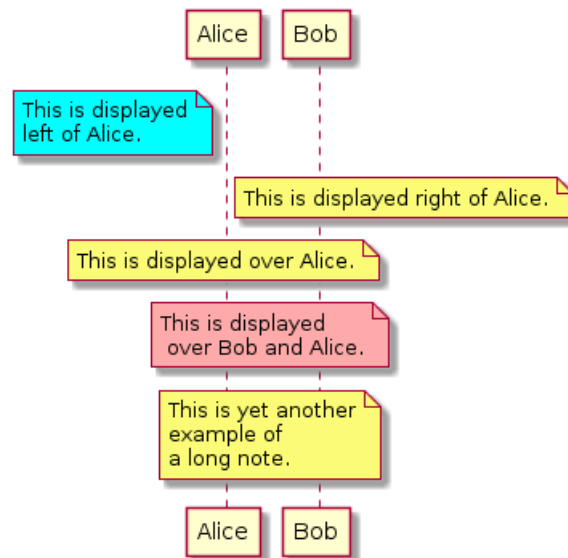
Bob->>Alice : ok
note right: this is another note

Bob->>Bob : I am thinking
note left
```

```

a note
can also be defined
on several lines
end note
@enduml

```



1.14 其他的注释

可以使用 `note left of`, `note right of` 或 `note over` 在节点 (participant) 的相对位置放置注释。

还可以通过修改背景色来高亮显示注释。

以及使用关键字 `end note` 来添加多行注释。

```

@startuml
participant Alice
participant Bob
note left of Alice #aqua
This is displayed
left of Alice.
end note

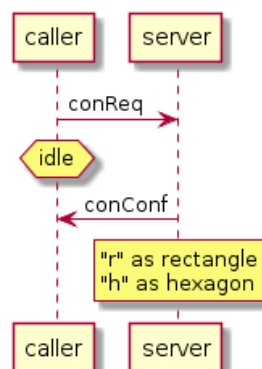
note right of Alice: This is displayed right of Alice.

note over Alice: This is displayed over Alice.

note over Alice, Bob #FFAAAA: This is displayed\n over Bob and Alice.

note over Bob, Alice
This is yet another
example of
a long note.
end note
@enduml

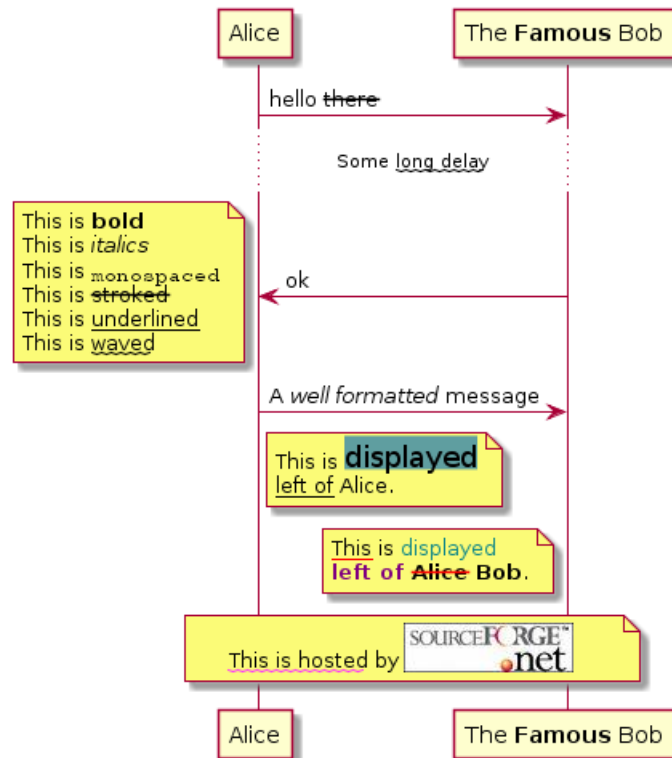
```



1.15 改变备注框的形状

你可以使用 `hnote` 和 `rnote` 这两个关键字来修改备注框的形状。

```
@startuml
caller -> server : conReq
hnote over caller : idle
caller <- server : conConf
rnote over server
"r" as rectangle
"h" as hexagon
endrnote
@enduml
```



1.16 Creole 和 HTML

可以使用 creole 格式。

```
@startuml
participant Alice
participant "The **Famous** Bob" as Bob

Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
This is **bold**
This is //italics//
This is ""monospaced""
This is --stroked--
This is __underlined__
This is ~~waved~~
end note

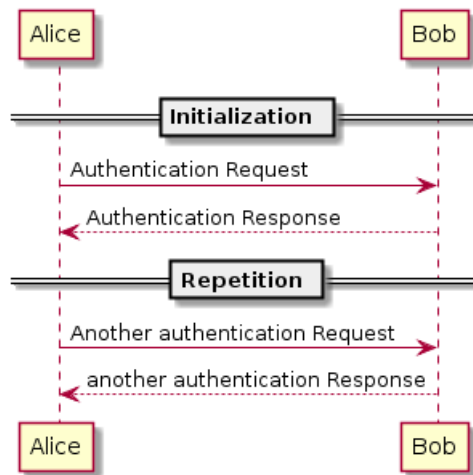
Alice -> Bob : A //well formatted// message
note right of Alice
This is <back:cadetblue><size:18>displayed</size></back>
__left of__ Alice.
end note
note left of Bob
<u:red>This</u> is <color #118888>displayed</color>
```



```

**<color purple>left of</color> <s:red>Alice</strike> Bob**.
end note
note over Alice, Bob
<w:#FF33FF>This is hosted</w> by <img sourceforge.jpg>
end note
@enduml

```



1.17 分隔符

你可以通过使用 == 关键词来将你的图表分割多个步骤。

```

@startuml

== Initialization ==

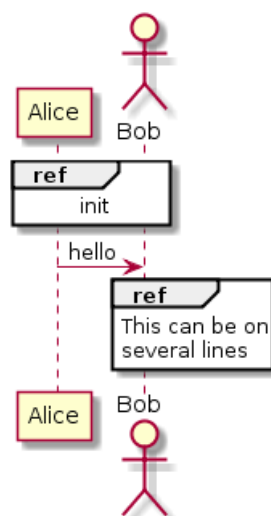
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

== Repetition ==

Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response

@enduml

```



1.18 引用

You can use reference in a diagram, using the keyword **ref** over.

```

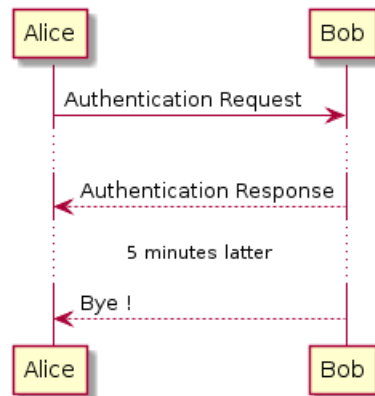
@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

ref over Bob
This can be on
several lines
end ref
@enduml

```



1.19 延迟

你可以使用... 来表示延迟，并且还可以给延迟添加注释。

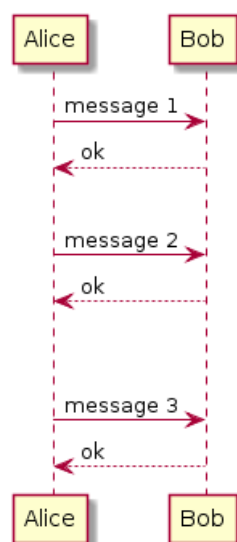
```

@startuml

Alice -> Bob: Authentication Request
...
Bob --> Alice: Authentication Response
...5 minutes latter...
Bob --> Alice: Bye !

@enduml

```



1.20 空间

你可以使用 ||| 来增加空间。

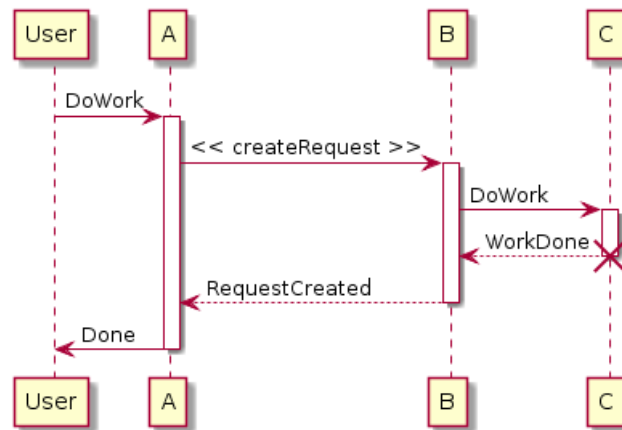


还可以使用数字指定增加的像素的数量。

```
@startuml
```

```
Alice -> Bob: message 1
Bob --> Alice: ok
|||
Alice -> Bob: message 2
Bob --> Alice: ok
||45||
Alice -> Bob: message 3
Bob --> Alice: ok
```

```
@enduml
```



1.21 生命线的激活与撤销

关键字 **activate** 和 **deactivate** 用来表示参与者的生命活动。

一旦参与者被激活，它的生命线就会显示出来。

activate 和 **deactivate** 适用于以上情形。

destroy 表示一个参与者的生命线的终结。

```
@startuml
```

```
participant User
```

```
User -> A: DoWork
activate A
```

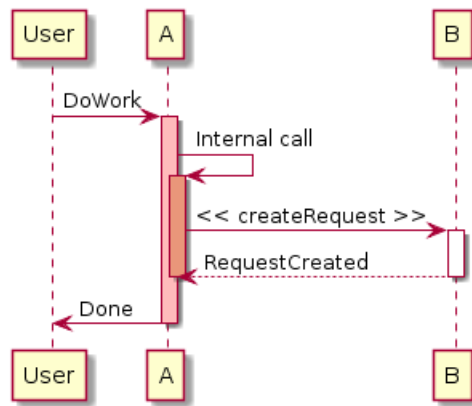
```
A -> B: << createRequest >>
activate B
```

```
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C
```

```
B --> A: RequestCreated
deactivate B
```

```
A -> User: Done
deactivate A
```

```
@enduml
```



还可以使用嵌套的生命线，并且运行给生命线添加颜色。

```

@startuml
participant User

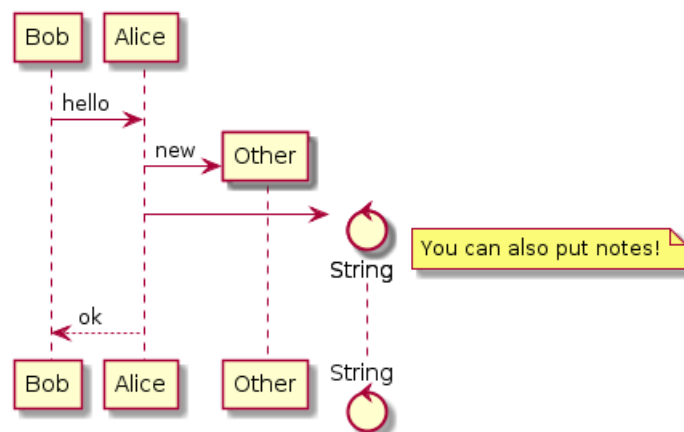
User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

A -> B: << createRequest >>
activate B

B --> A: RequestCreated
deactivate B
deactivate A
A -> User: Done
deactivate A

@enduml
  
```



1.22 创建参与者

你可以把关键字 `create` 放在第一次接收到消息之前，以强调本次消息实际上是在创建新的对象。

```

@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

create control String
Alice -> String
note right : You can also put notes!
  
```

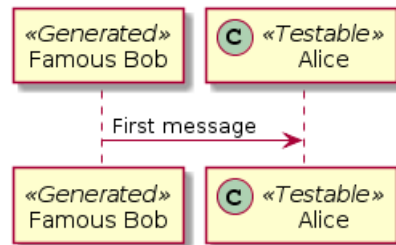


```

Bob ->]
Bob ->o]
Bob o->o]
Bob ->x]

Bob <-]
Bob x<-]
@enduml

```



1.24 构造类型和圈点

可以使用 << 和 >> 给参与者添加构造类型。

在构造类型中，你可以使用 (X,color) 格式的语法添加一个圆圈圈起来的字符。

```

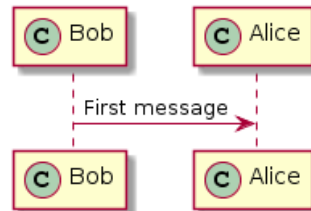
@startuml

participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>

Bob->>Alice: First message

@enduml

```



By default, the *guillemet* character is used to display the stereotype. You can change this behaviour using the skinparam *guillemet*:

```

@startuml

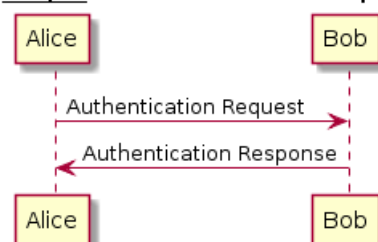
skinparam guillemet false
participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>

Bob->>Alice: First message

@enduml

```

Simple communication example



```

@startuml
participant Bob << (C,#ADD1B2) >>
participant Alice << (C,#ADD1B2) >>

Bob->>Alice: First message

@enduml

```

**Simple communication example
on several lines**



1.25 更多标题信息

你可以在标题中使用 creole 格式。

```

@startuml

title __Simple__ **communication** example

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml

```

**Simple communication example
on several lines and using **html****



在标题描述中使用 \n 表示换行。

```

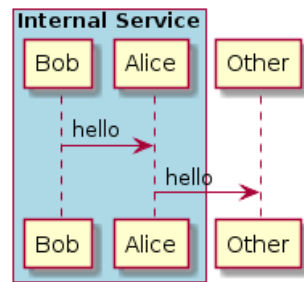
@startuml

title __Simple__ communication example\non several lines

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml

```



还可以使用关键字 `title` 和 `end title` 定义多行标题。

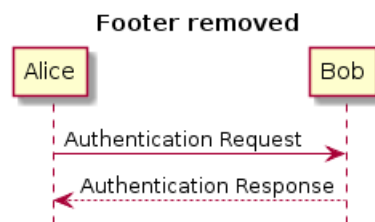
```

@startuml

title
<u>Simple</u> communication example
on <i>several</i> lines and using <font color=red>html</font>
This is hosted by <img:sourceforge.jpg>
end title

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml
  
```



1.26 包裹参与者

可以使用 `box` 和 `end box` 画一个盒子将参与者包裹起来。

还可以在 `box` 关键字之后添加标题或者背景颜色。

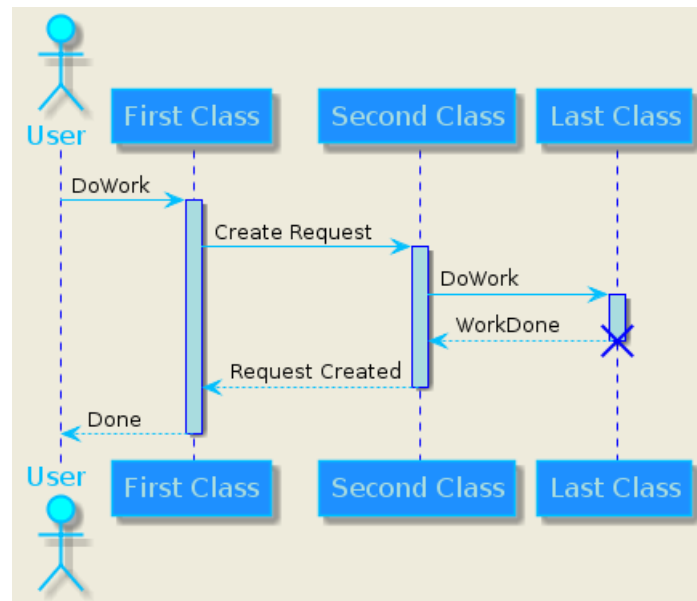
```

@startuml

box "Internal Service" #LightBlue
participant Bob
participant Alice
end box
participant Other

Bob -> Alice : hello
Alice -> Other : hello

@enduml
  
```



1.27 移除脚注

使用 `hide footbox` 关键字移除脚注。

```

@startuml

hide footbox
title Footer removed

Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

@enduml

```

1.28 外观参数 (skinparam)

使用 `skinparam` 命令改变颜色和字体。

如下场景可以使用这一命令：

- 在图示定义中，
- 在一个包含文件中，
- 在由命令行或者 ANT 任务提供的配置文件中。

```

@startuml
skinparam backgroundColor #EEEEBD

skinparam sequence {
  ArrowColor DeepSkyBlue
  ActorBorderColor DeepSkyBlue
  LifeLineBorderColor blue
  LifeLineBackgroundColor #A9DCDF

  ParticipantBorderColor DeepSkyBlue
  ParticipantBackgroundColor DodgerBlue
  ParticipantFontName Impact
  ParticipantFontSize 17
  ParticipantFontColor #A9DCDF

  ActorBackgroundColor aqua
  ActorFontColor DeepSkyBlue
  ActorFontSize 17
  ActorFontName Aapex
}

```

```
}  
  
actor User  
participant "First Class" as A  
participant "Second Class" as B  
participant "Last Class" as C  
  
User -> A: DoWork  
activate A  
  
A -> B: Create Request  
activate B  
  
B -> C: DoWork  
activate C  
C --> B: WorkDone  
destroy C  
  
B --> A: Request Created  
deactivate B  
  
A --> User: Done  
deactivate A  
  
@enduml
```

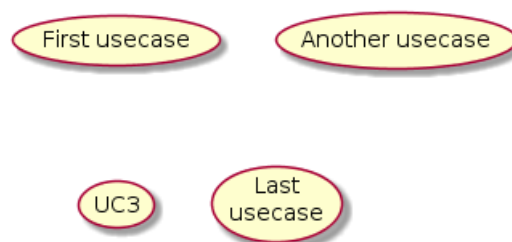
2 用例图

2.1 用例

用例用圆括号括起来。

也可以用关键字 `usecase` 来定义用例。还可以用关键字 `as` 定义一个别名，这个别名可以在以后定义关系的时候使用。

```
@startuml
(First usecase)
(Another usecase) as (UC2)
usecase UC3
usecase (Last\nusecase) as UC4
@enduml
```



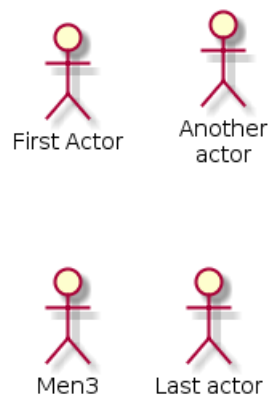
2.2 角色

角色用两个冒号包裹起来。

也可以用 `actor` 关键字来定义角色。还可以用关键字 `as` 来定义一个别名，这个别名可以在以后定义关系的时候使用。

后面我们会看到角色的定义是可选的。

```
@startuml
:First Actor:
:Another\nactor: as Men2
actor Men3
actor :Last actor: as Men4
@enduml
```



2.3 用例描述

如果想定义跨越多行的用例描述，可以用双引号将其裹起来。

还可以使用这些分隔符：`--` `..` `==` `__`。并且还可以在分隔符中间放置标题。

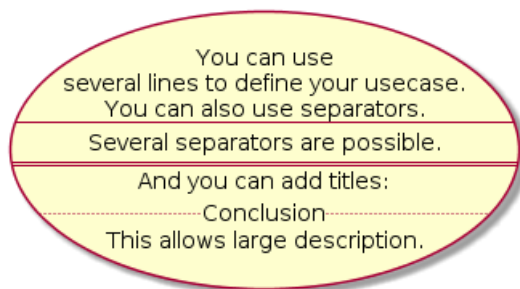
```

@startuml

usecase UC1 as "You can use
several lines to define your usecase.
You can also use separators.
--
Several separators are possible.
==
And you can add titles:
..Conclusion..
This allows large description."

@enduml

```



2.4 基础示例

用箭头 --> 连接角色和用例。

横杠 "-" 越多，箭头越长。通过在箭头定义的后面加一个冒号及文字的方式来添加标签。

在这个例子中，*User* 并没有定义，而是直接拿来当做一个角色使用。

```

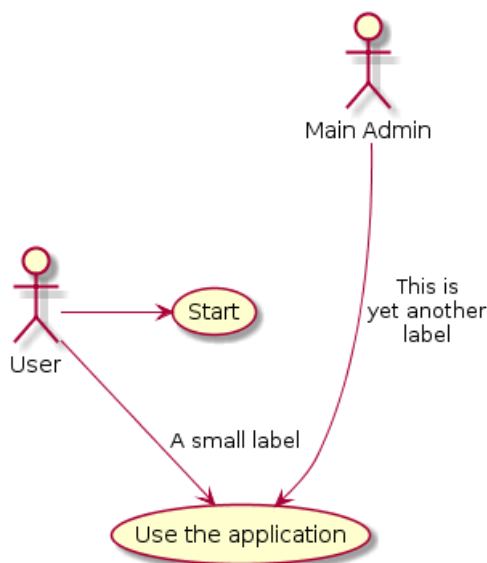
@startuml

User -> (Start)
User --> (Use the application) : A small label

:Main Admin: ---> (Use the application) : This is yet another label

@enduml

```



2.5 继承

如果一个角色或者用例继承于另一个，那么可以用 <|-- 符号表示（它表示 ）。

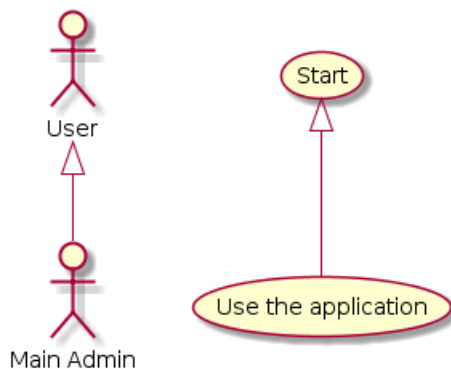
```

@startuml
:Main Admin: as Admin
(Use the application) as (Use)

User <|-- Admin
(Start) <|-- (Use)

@enduml

```



2.6 使用注释

可以用 `note left of` , `note right of` , `note top of` , `note bottom of` 等关键字给一个对象添加注释。

注释还可以通过 `note` 关键字来定义, 然后用 `..` 连接其他对象。

```

@startuml
:Main Admin: as Admin
(Use the application) as (Use)

User -> (Start)
User --> (Use)

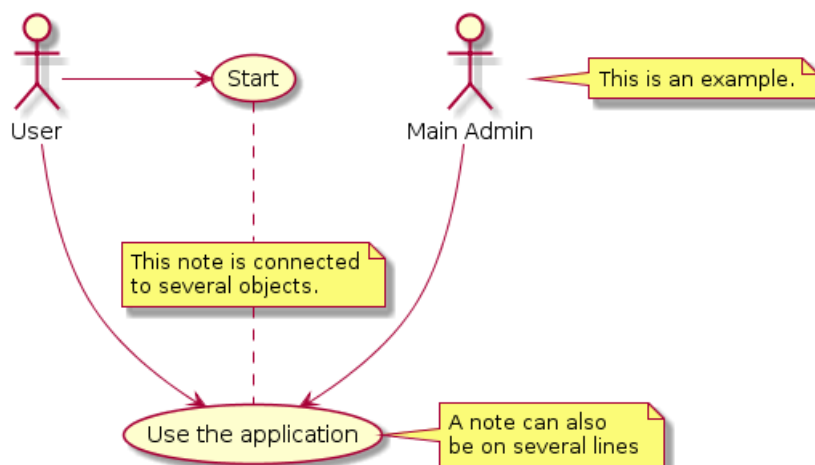
Admin ---> (Use)

note right of Admin : This is an example.

note right of (Use)
A note can also
be on several lines
end note

note "This note is connected\nto several objects." as N2
(Start) .. N2
N2 .. (Use)
@enduml

```



2.7 构造类型

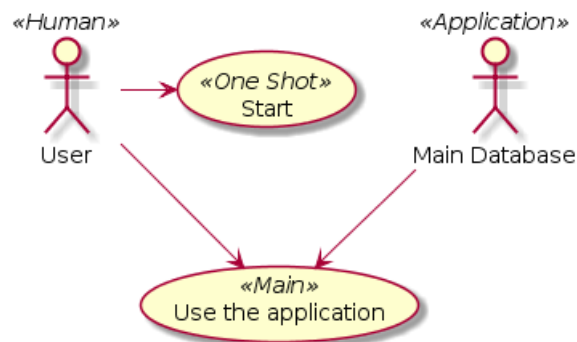
用” << ” 和” >> ” 来定义角色或者用例的构造类型。

```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

User -> (Start)
User --> (Use)

MySql --> (Use)

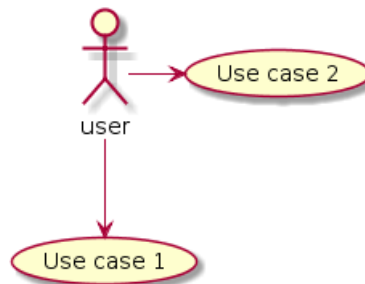
@enduml
```



2.8 改变箭头方向

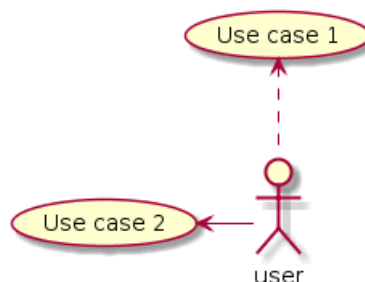
默认连接是竖直方向的，用 --表示，可以用一个横杠或点来表示水平连接。

```
@startuml
:user: --> (Use case 1)
:user: -> (Use case 2)
@enduml
```



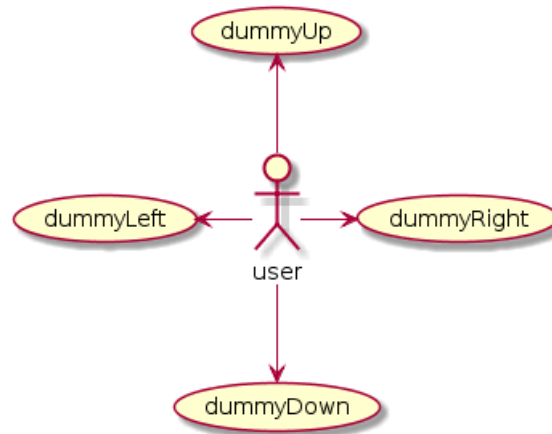
也可以通过翻转箭头来改变方向。

```
@startuml
(Use case 1) <.. :user:
(Use case 2) <- :user:
@enduml
```



还可以通过给箭头添加 `left`, `right`, `up` 或 `down` 等关键字来改变方向。

```
@startuml
: user: -left-> (dummyLeft)
: user: -right-> (dummyRight)
: user: -up-> (dummyUp)
: user: -down-> (dummyDown)
@enduml
```



这些方向关键字也可以只是用首字母或者前两个字母的缩写来代替。但是请注意，这样的缩写不要乱用，Graphviz 不喜欢这样。

2.9 添加标题

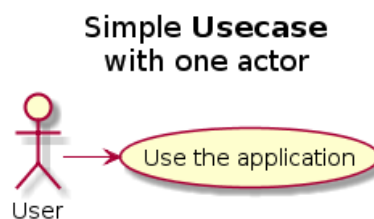
用 `title` 关键字来添加标题。

还可以用 `title` 和 `end title` 来定义长标题。

```
@startuml
title Simple <b>Usecase</b>\nwith one actor

"Use the application" as (Use)
User -> (Use)

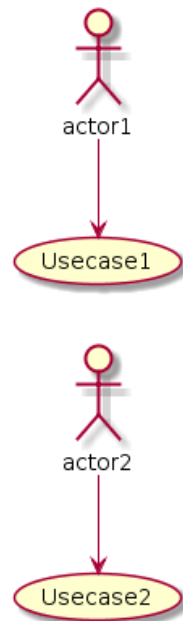
@enduml
```



2.10 分割图示

用 `newpage` 关键字将图示分解为多个页面。

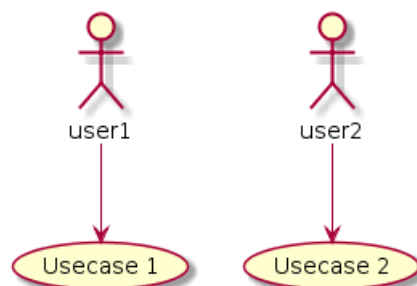
```
@startuml
: actor1: --> (Usecase1)
newpage
: actor2: --> (Usecase2)
@enduml
```



2.11 从左向右方向

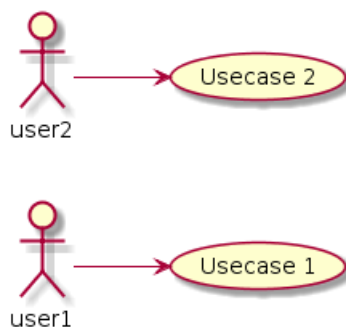
默认从上往下构建图示。

```
@startuml
'default
top to bottom direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)
@enduml
```



你可以用 `left to right direction` 命令改变图示方向。

```
@startuml
left to right direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)
@enduml
```



2.12 显示参数

用 `skinparam` 改变字体和颜色。
可以在如下场景中使用：

- 在图示的定义中，
- 在引入的文件中，
- 在命令行或者 ANT 任务提供的配置文件中。

你也可以给构造的角色和用例指定特殊颜色和字体。

```
@startuml
skinparam handwritten true

skinparam usecase {
  BackgroundColor DarkSeaGreen
  BorderColor DarkSlateGray

  BackgroundColor<< Main >> YellowGreen
  BorderColor<< Main >> YellowGreen

  ArrowColor Olive
  ActorBorderColor black
  ActorFontName Courier

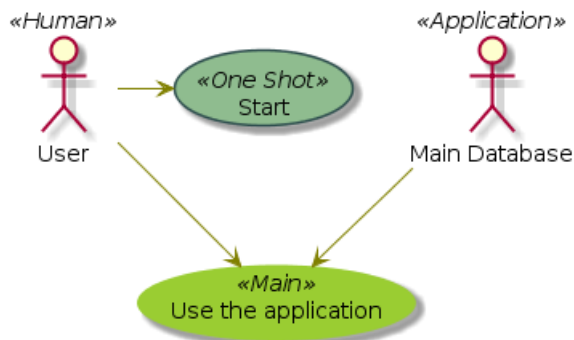
  ActorBackgroundColor<< Human >> Gold
}

User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

User -> (Start)
User --> (Use)

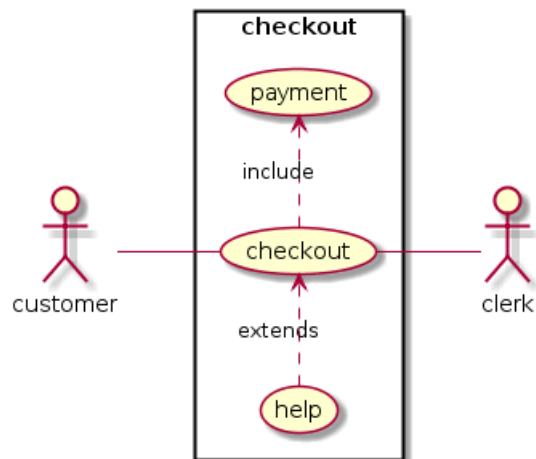
MySql --> (Use)

@enduml
```



2.13 一个完整的例子




```
@startuml
left to right direction
skinparam packageStyle rect
actor customer
actor clerk
rectangle checkout {
customer -- (checkout)
(checkout) .> (payment) : include
(help) .> (checkout) : extends
(checkout) -- clerk
}
@enduml
```



3 类图

3.1 类之间的关系

类之间的关系通过下面的符号定义:

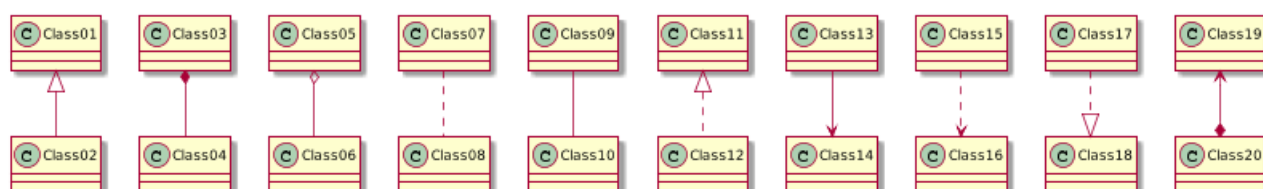
继承 (extension)	< --	
合成 (composition)	*--	
聚合 (aggregation)	o--	

使用“..”来代替“--”可以得到点线.

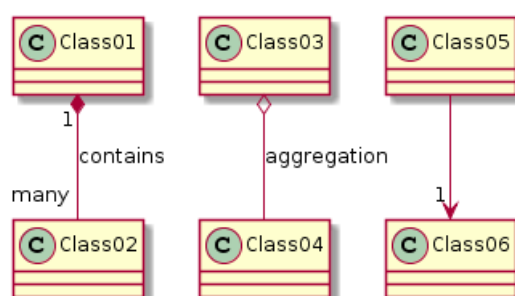
在这些规则下, 也可以绘制下列图形

Knowing those rules, it is possible to draw the following drawings:

```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
Class09 -- Class11
Class13 --> Class14
Class15 -.-> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```



```
@startuml
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```

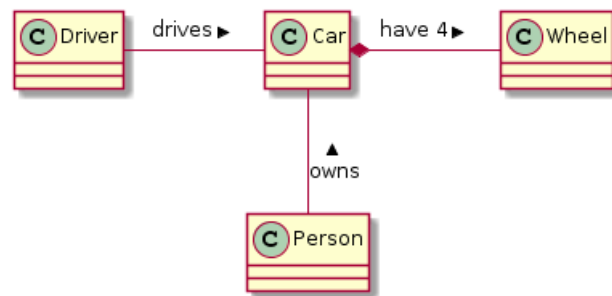


3.2 关系上的标识

在关系之间使用标签来说明时, 使用“:”后接标签文字。

对元素的说明, 你可以在每一边使用“”来说明.

```
@startuml
Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : aggregation
Class05 --> "1" Class06
@enduml
```



在标签的开始或结束位置添加 < 或 > 以表明是哪个对象作用到哪个对象上。

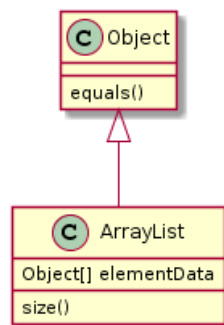
```

@startuml
class Car

Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns

@enduml

```



3.3 添加方法

为了声明域或者方法，你可以使用后接域名或方法名。

系统检查是否有括号来判断是方法还是域。

```
@startuml
Object <|-- ArrayList

Object : equals()
ArrayList : Object[] elementData
ArrayList : size()

@enduml
```



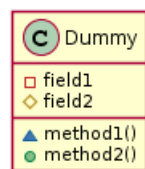
也可以使用 { 把域或者方法括起来

注意，这种语法对于类型/名字的顺序是非常灵活的。

```
@startuml
class Dummy {
String data
void methods()
}

class Flight {
flightNumber : Integer
departureTime : Date
}

@enduml
```

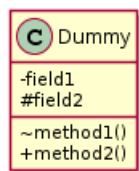


3.4 定义可访问性

一旦你定义了域或者方法，你可以定义相应条目的可访问性质。

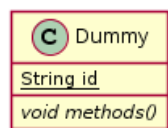
-	□	private
#	◇	protected
~	△	package private
+	○	public

```
@startuml
class Dummy {
-field1
#field2
~method1()
+method2()
}
@enduml
```



你可以采用以下命令停用这些特性 `skinparam classAttributeIconSize 0` :

```
@startuml
skinparam classAttributeIconSize 0
class Dummy {
-field1
#field2
~method1()
+method2()
}
@enduml
```

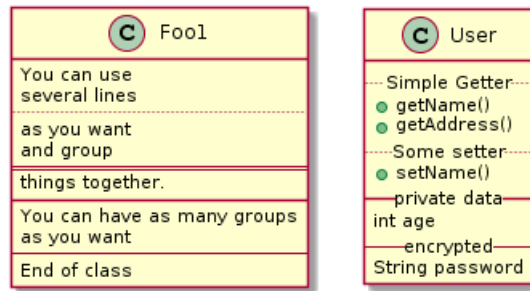


3.5 抽象与静态

通过修饰符 `static` 或者 `abstract`，可以定义静态或者抽象的方法或者属性。

这些修饰符可以写在行的开始或者结束。也可以使用 `classifier` 这个修饰符来代替 `static`。

```
@startuml
class Dummy {
{static} String id
{abstract} void methods()
}
@enduml
```



3.6 高级类体

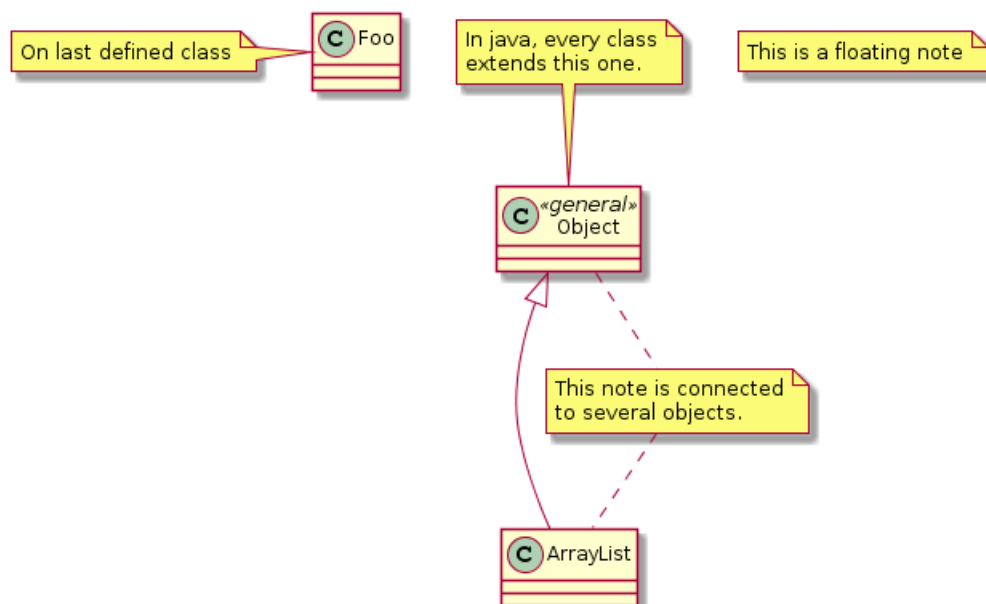
PlantUML 默认自动将方法和属性重新分组，你可以自己定义分隔符来重排方法和属性，下面的分隔符都是可用的：-- .. == __.

还可以在分隔符中添加标题：

```
@startuml
class Foo1 {
You can use
several lines
..
as you want
and group
==
things together.
--
You can have as many groups
as you want
--
End of class
}

class User {
.. Simple Getter ..
+ getName()
+ getAddress()
.. Some setter ..
+ setName()
__ private data __
int age
-- encrypted --
String password
}

@enduml
```



3.7 备注和模板

模板通过类关键字 (“«” 和”»”) 来定义

你可以使用 `note left of`, `note right of`, `note top of`, `note bottom of` 这些关键字来添加备注。

你还可以在类的声明末尾使用 `note left`, `note right`, `note top`, `note bottom` 来添加备注。

此外，单独用 `note` 这个关键字也是可以的，使用 `..` 符号可以作出一条连接它与其它对象的虚线。

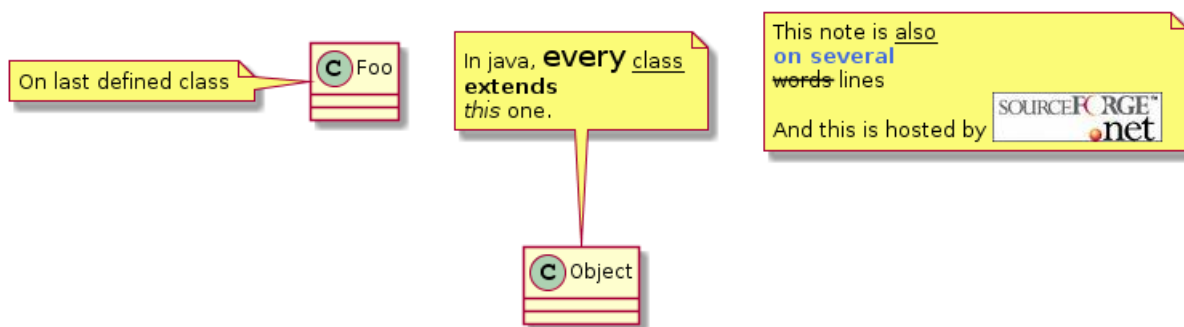
```
@startuml
class Object << general >>
Object <|--- ArrayList

note top of Object : In java, every class\nextends this one.

note "This is a floating note" as N1
note "This note is connected\nto several objects." as N2
Object .. N2
N2 .. ArrayList

class Foo
note left: On last defined class

@enduml
```



3.8 更多注释

可以在注释中使用部分 html 标签:

- ``
- `<u>`
- `<i>`
- `<s>`, ``, `<strike>`
- `` or ``
- `<color:#AAAAAA>` or `<color:colorName>`
- `<size:nn>` to change font size
- `` or `<img:file>` : the file must be accessible by the filesystem

You can also have a note on several lines You can also define a note on the last defined class using `note left`, `note right`, `note top`, `note bottom`.

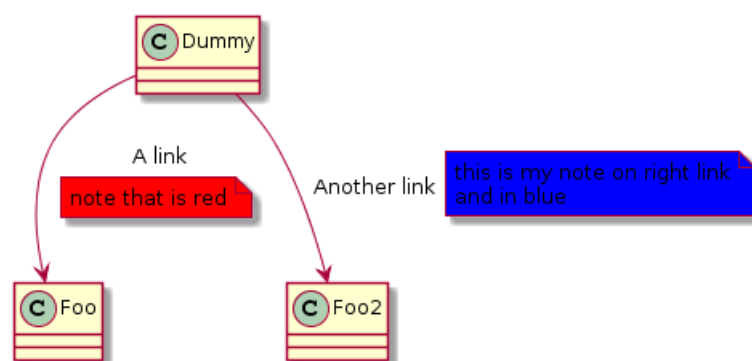
```
@startuml

class Foo
note left: On last defined class

note top of Object
In java, <size:18>every</size> <u>class</u>
<b>extends</b>
<i>this</i> one.
end note

note as N1
This note is <u>also</u>
<b><color:royalBlue>on several</color>
<s>words</s> lines
And this is hosted by <img:sourceforge.jpg>
end note

@enduml
```



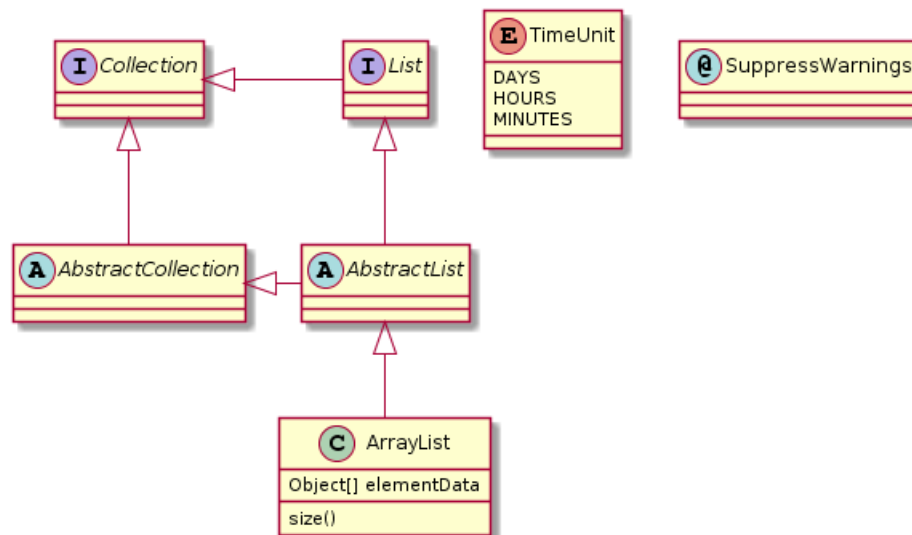
3.9 Note on links

It is possible to add a note on a link, just after the link definition, using `note on link`.

You can also use `note left on link`, `note right on link`, `note top on link`, `note bottom on link` if you want to change the relative position of the note with the label.

```
@startuml
class Dummy
Dummy --> Foo : A link
note on link #red: note that is red

Dummy --> Foo2 : Another link
note right on link #blue
this is my note on right link
and in blue
end note
@enduml
```



3.10 抽象类和接口

用关键字"abstract" 或"abstract class" 来定义抽象类。抽象类用斜体显示。也可以使用 interface, annotation 和 enum 关键字。

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

List <|-- AbstractList
Collection <|-- AbstractCollection

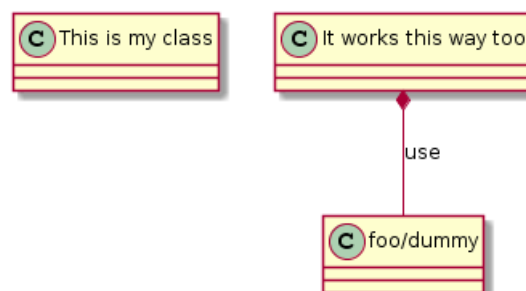
Collection <|-- List
AbstractCollection <|-- AbstractList
AbstractList <|-- ArrayList

class ArrayList {
Object[] elementData
size()
}

enum TimeUnit {
DAYS
HOURS
MINUTES
}

annotation SuppressWarnings

@enduml
```



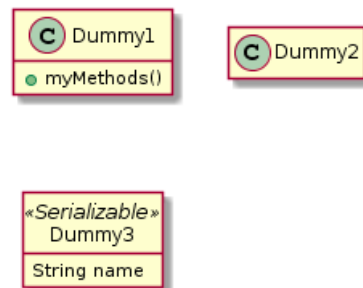
3.11 Using non-letters

If you want to use non-letters in the class (or enum...) display, you can either :

- Use the `as` keyword in the class definition
- Put quotes `"` around the class name

```
@startuml
class "This is my class" as class1
class class2 as "It works this way too"

class2 *-- "foo/dummy" : use
@enduml
```



3.12 Hide attributes, methods...

You can parameterize the display of classes using the **hide/show** command.

The basic command is: **hide empty members**. This command will hide attributes or methods if they are empty.

Instead of **empty members**, you can use:

- **empty fields** or **empty attributes** for empty fields,
- **empty methods** for empty methods,
- **fields** or **attributes** which will hide fields, even if they are described,
- **methods** which will hide methods, even if they are described,
- **members** which will hide fields and methods, even if they are described,
- **circle** for the circled character in front of class name,
- **stereotype** for the stereotype.

You can also provide, just after the **hide** or **show** keyword:

- **class** for all classes,
- **interface** for all interfaces,
- **enum** for all enums,
- **<<foo1>>** for classes which are stereotyped with *foo1*,
- an existing class name.

You can use several **show/hide** commands to define rules and exceptions.

```
@startuml

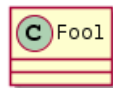
class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

hide members
hide <<Serializable>> circle
show Dummy1 methods
show <<Serializable>> fields

@enduml
```



3.13 Hide classes

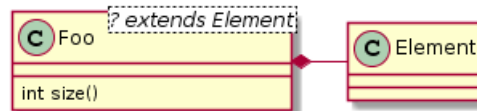
You can also use the **show/hide** commands to hide classes.

This may be useful if you define a large !included file, and if you want to hide some classes after file inclusion.

```

@startuml
class Foo1
class Foo2
Foo2 *-- Foo1
hide Foo2
@enduml

```



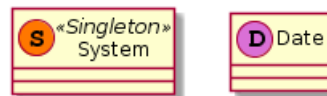
3.14 Use generics

You can also use bracket < and > to define generics usage in a class.

```

@startuml
class Foo<? extends Element> {
int size()
}
Foo *-- Element
@enduml

```



3.15 Specific Spot

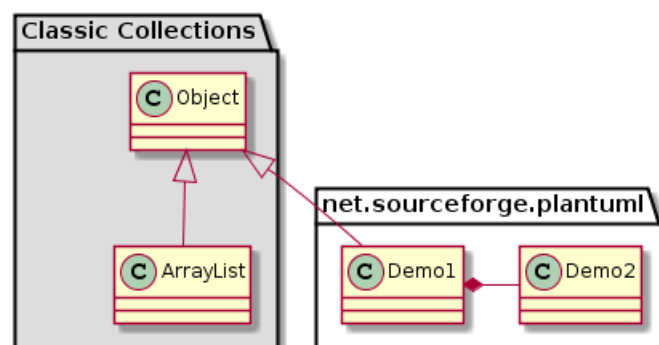
Usually, a spotted character (C, I, E or A) is used for classes, interface, enum and abstract classes.

But you can define your own spot for a class when you define the stereotype, adding a single character and a color, like in this example:

```

@startuml
class System << (S,#FF7700) Singleton >>
class Date << (D,orchid) >>
@enduml

```



3.16 Packages

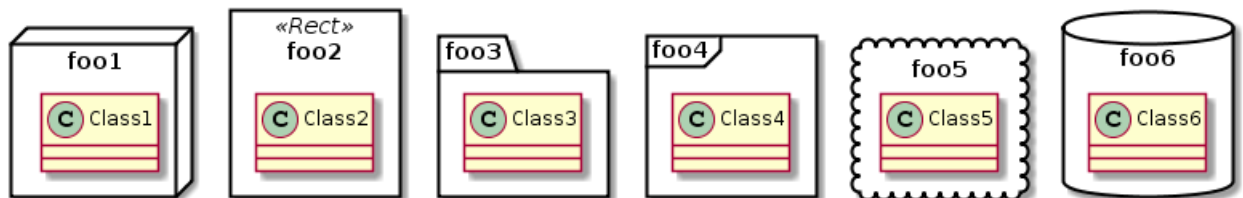
You can define a package using the **package** keyword, and optionally declare a background color for your package (Using a html color code or name).

Note that package definitions can be nested.

```
@startuml
package "Classic Collections" #DDDDDD {
Object <|-- ArrayList
}

package net.sourceforge.plantuml {
Object <|-- Demo1
Demo1 *- Demo2
}

@enduml
```



3.17 Packages style

There are different styles available for packages.

You can specify them either by setting a default style with the command: `skinparam packageStyle`, or by using a stereotype on the package:

```
@startuml
scale 750 width
package foo1 <<Node>> {
class Class1
}

package foo2 <<Rect>> {
class Class2
}

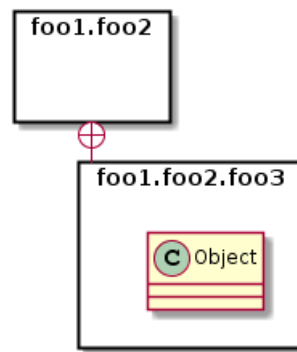
package foo3 <<Folder>> {
class Class3
}

package foo4 <<Frame>> {
class Class4
}

package foo5 <<Cloud>> {
class Class5
}

package foo6 <<Database>> {
class Class6
}

@enduml
```



You can also define links between packages, like in the following example:

```

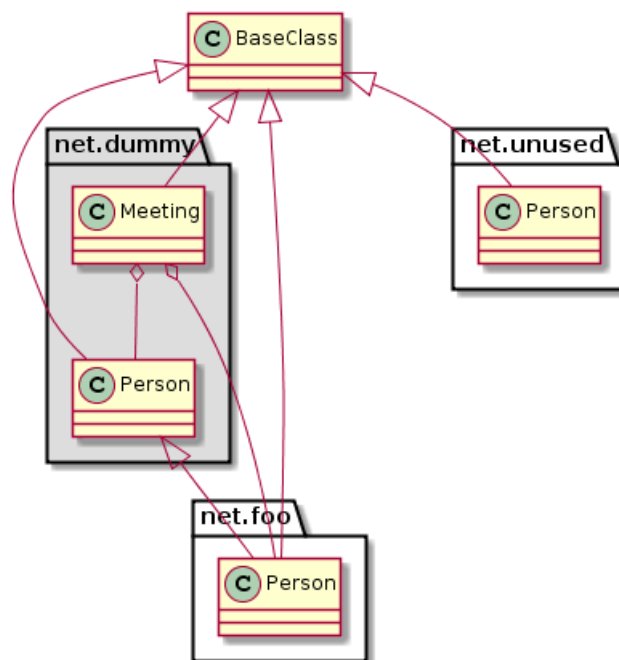
@startuml
    skinparam packageStyle rect

    package foo1.foo2 {
    }

    package foo1.foo2.foo3 {
        class Object
    }

    foo1.foo2 +-- foo1.foo2.foo3

@enduml
    
```



3.18 Namespaces

In packages, the name of a class is the unique identifier of this class. It means that you cannot have two classes with the very same name in different packages.

In that case, you should use namespaces instead of packages.

You can refer to classes from other namespaces by fully qualify them. Classes from the default namespace are qualified with a starting dot.

Note that you don't have to explicitly create namespace : a fully qualified class is automatically put in the right namespace.

```

@startuml

class BaseClass

namespace net.dummy #DDDDDD {
  .BaseClass <|-- Person
  Meeting o-- Person

  .BaseClass <|-- Meeting
}

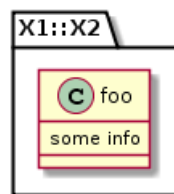
namespace net.foo {
  net.dummy.Person <|-- Person
  .BaseClass <|-- Person
}

net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml

```



3.19 Automatic namespace creation

You can define another separator (other than the dot) using the command: `set namespaceSeparator ???`.

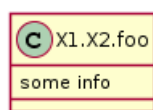
```

@startuml

set namespaceSeparator ::
class X1::X2::foo {
  some info
}

@enduml

```



You can disable automatic package creation using the command `set namespaceSeparator none`.

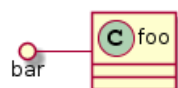
```

@startuml

set namespaceSeparator none
class X1.X2.foo {
  some info
}

@enduml

```

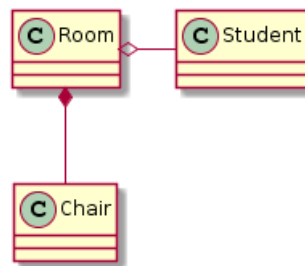


3.20 Lollipop interface

You can also define lollipop interface on classes, using the following syntax:

- `bar ()- foo`
- `bar ()-- foo`
- `foo -() bar`

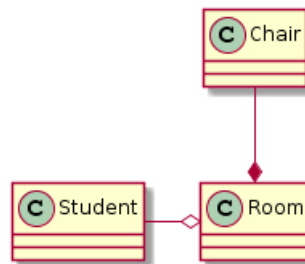
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 Changing arrows direction

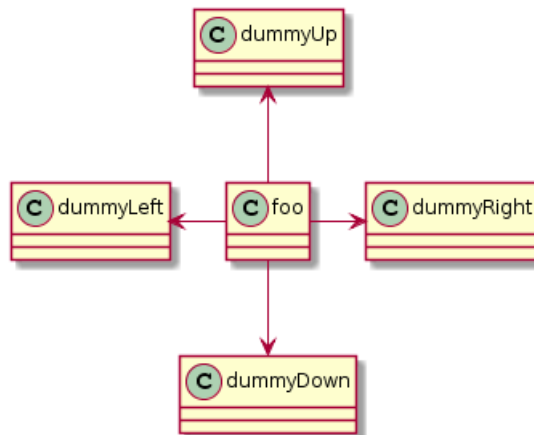
默认类之间采用两个破折号 `--` 显示出垂直方向的线. 要得到水平方向的可以像这样使用单破折号 (或者点):

```
@startuml
Room o- Student
Room *-- Chair
@enduml
```



You can also change directions by reversing the link:

```
@startuml
Student -o Room
Chair --* Room
@enduml
```

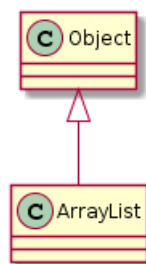


It is also possible to change arrow direction by adding `left`, `right`, `up` or `down` keywords inside the arrow:

```

@startuml
foo -left-> dummyLeft
foo -right-> dummyRight
foo -up-> dummyUp
foo -down-> dummyDown
@enduml
  
```

Simple example of title



You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

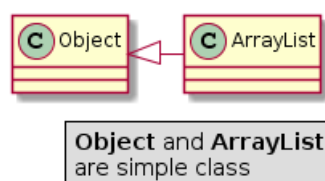
3.22 Title the diagram

The `title` keyword is used to put a title.

You can use `title` and `end title` keywords for a longer title, as in sequence diagrams.

```

@startuml
title Simple <b>example</b>\nof title
Object <|-- ArrayList
@enduml
  
```



3.23 Legend the diagram

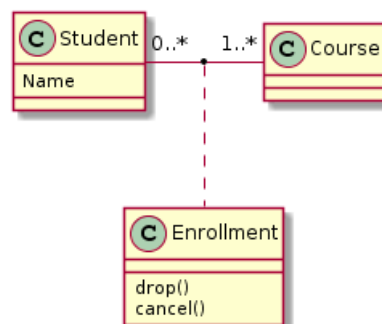
The `legend` and `end legend` are keywords is used to put a legend.

You can optionally specify to have `left`, `right` or `center` alignment for the legend.

```
@startuml
Object <|- ArrayList

legend right
<b>Object</b> and <b>ArrayList</b>
are simple class
endlegend

@enduml
```

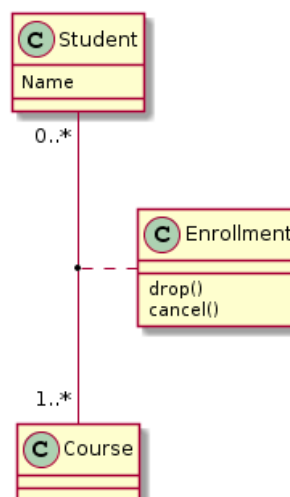


3.24 Association classes

You can define *association class* after that a relation has been defined between two classes, like in this example:

```
@startuml
class Student {
Name
}
Student "0..*" -- "1..*" Course
(Student, Course) .. Enrollment

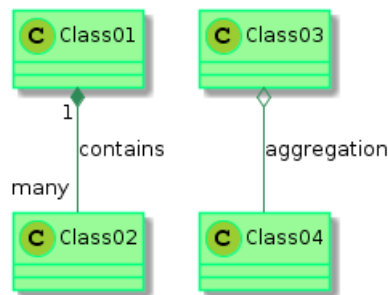
class Enrollment {
drop()
cancel()
}
@enduml
```



You can define it in another direction:

```
@startuml
class Student {
Name
}
Student "0..*" -- "1..*" Course
(Student, Course) . Enrollment

class Enrollment {
drop()
cancel()
}
@enduml
```



3.25 Skinparam

You can use the `skinparam` command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

```
@startuml
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : aggregation
@enduml
```

```
[From /home/ec2-user/database/tmp/ZH/classes.tex (line 1116) ]
... (skipping 12 lines) ...
Class01 << Foo >>
Class01 "1" *-- "many" Class02 : contains
Class03<<Foo>> o-- Class04 : aggregation
Syntax Error?
```

3.26 Skinned Stereotypes

You can define specific color and fonts for stereotyped classes.



```

@startuml

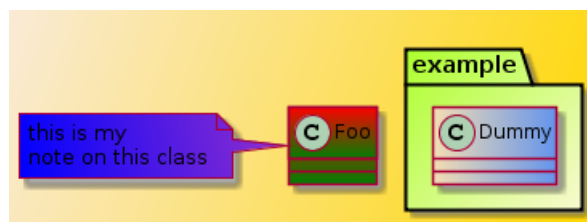
skinparam class {
  BackgroundColor PaleGreen
  ArrowColor SeaGreen
  BorderColor SpringGreen
  BackgroundColor<<Foo>> Wheat
  BorderColor<<Foo>> Tomato
}
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

Class01 <<Foo>>
Class03 <<Foo>>
Class01 "1" *-- "many" Class02 : contains

Class03 o-- Class04 : aggregation

@enduml

```



3.27 Color gradient

It's possible to declare individual color for classes or note using the notation.

You can use either standard color name or RGB code.

You can also use color gradient in background, with the following syntax: two colors names separated either by:

- |,
- /,
- \,
- or -

depending the direction of the gradient.

For example, you could have:

```

@startuml

skinparam backgroundcolor AntiqueWhite/Gold
skinparam classBackgroundColor Wheat|CornflowerBlue

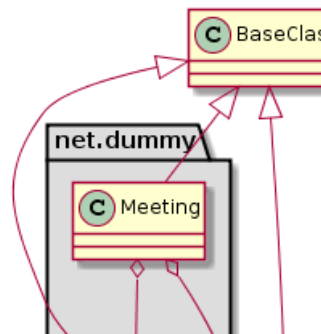
class Foo #red-green
note left of Foo #blue\9932CC
this is my
note on this class
end note

package example #GreenYellow/LightGoldenRodYellow {
class Dummy
}

@enduml

```





3.28 Splitting large files

Sometimes, you will get some very large image files.

You can use the "page (hpages)x(vpages)" command to split the generated image into several files :

hpages is a number that indicated the number of horizontal pages, and **vpages** is a number that indicated the number of vertical pages.

```

@startuml
' Split into 4 pages
page 2x2

class BaseClass

namespace net.dummy #DDDDDD {
.BaseClass <|-- Person
Meeting o-- Person

.BaseClass <|-- Meeting
}

namespace net.foo {
net.dummy.Person <|-- Person
.BaseClass <|-- Person

net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person
@enduml

```

4 活动图

4.1 简单活动

使用 (*) 作为活动图的开始点和结束点。

有时，你可能想用 (*top) 强制开始点位于图示的顶端。

使用 --> 绘制箭头。

```
@startuml
(*) --> "First Activity"
"First Activity" --> (*)
@enduml
```

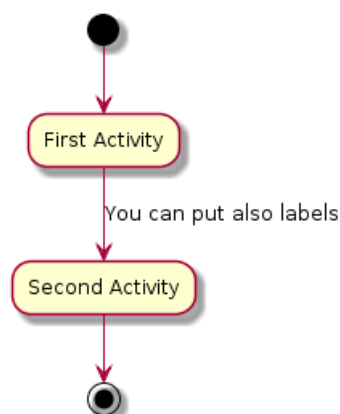


4.2 箭头上的标签

默认情况下，箭头开始于最接近的活动。

可以用 [和] 放在箭头定义的后面来添加标签。

```
@startuml
(*) --> "First Activity"
-->[You can put also labels] "Second Activity"
--> (*)
@enduml
```



4.3 改变箭头方向

你可以使用 -> 定义水平方向箭头，还可以使用下列语法强制指定箭头的方向：

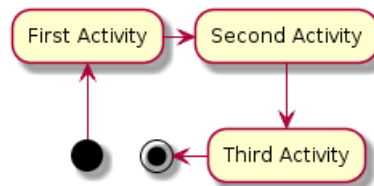
- -down-> (default arrow)
- -right-> or ->

```

    • -left->
    • -up->

@startuml
    (*) -up-> "First Activity"
    -right-> "Second Activity"
    --> "Third Activity"
    -left-> (*)
@enduml

```



4.4 分支

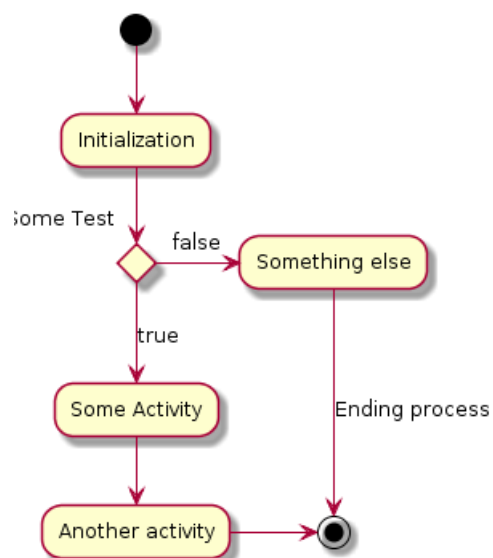
你可以使用关键字 `if/then/else` 创建分支。

```

@startuml
    (*) --> "Initialization"

    if "Some Test" then
        -->[true] "Some Activity"
        --> "Another activity"
        -right-> (*)
    else
        -->[false] "Something else"
        -->[Ending process] (*)
    endif
@enduml

```



不过，有时你可能需要重复定义同一个活动：

```

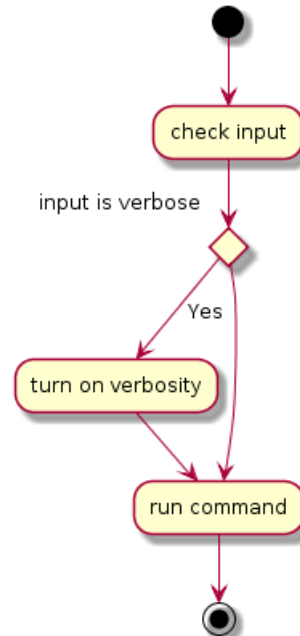
@startuml
    (*) --> "check input"
    If "input is verbose" then
        --> [Yes] "turn on verbosity"
    end

```

```

--> "run command"
else
--> "run command"
Endif
-->(*)
@enduml

```



4.5 更多分支

默认情况下，一个分支连接上一个最新的活动，但是也可以使用 `if` 关键字进行连接。还可以嵌套定义分支。

```

@startuml

(*) --> if "Some Test" then

-->[true] "activity 1"

if "" then
-> "activity 3" as a3
else
if "Other test" then
-left-> "activity 5"
else
--> "activity 6"
endif
endif

else

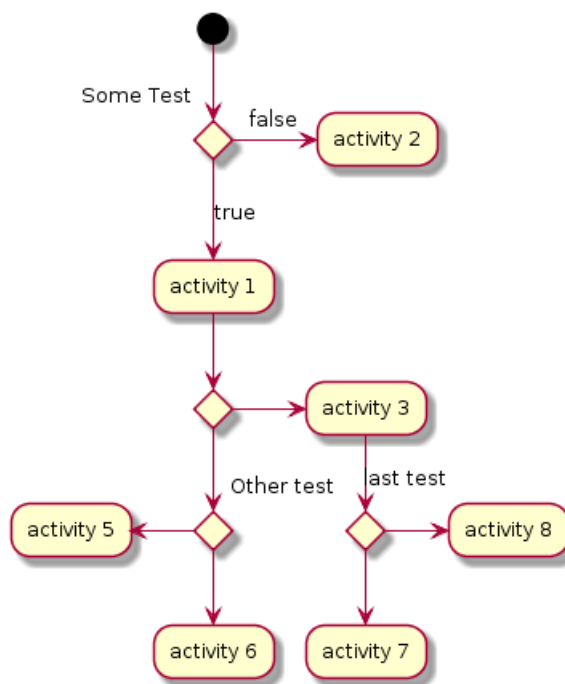
->[false] "activity 2"

endif

a3 --> if "last test" then
--> "activity 7"
else
-> "activity 8"
endif

@enduml

```



4.6 同步

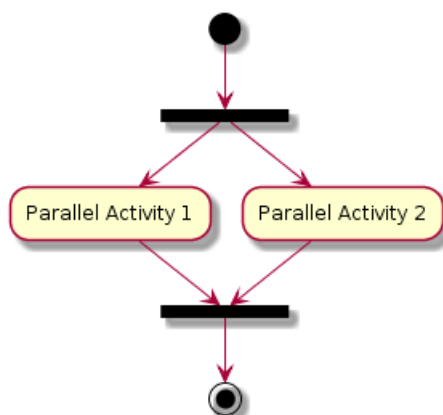
你可以使用“`=== code ===`”来显示同步条。

```

@startuml
(*) --> ===B1===
--> "Parallel Activity 1"
--> ===B2===

===B1=== --> "Parallel Activity 2"
--> ===B2===

--> (*)
@enduml
  
```



4.7 长的活动描述

定义活动时可以用 `\n` 来定义跨越多行的描述。

还可以用 `as` 关键字给活动起一个短的别名。这个别名可以在接下来的图示定义中使用。

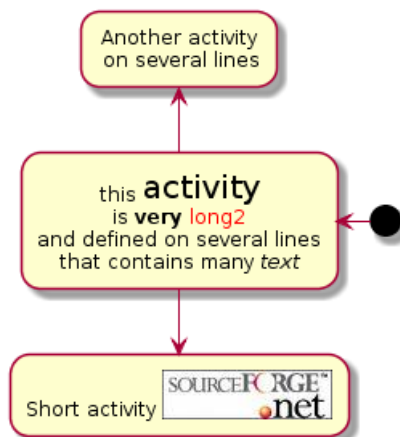
```

@startuml
(*) -left-> "this <size:20>activity</size>  
is <b>very</b> <color:red>long2</color>  
and defined on several lines  
that contains many <i>text</i>" as A1

-up-> "Another activity\n on several lines"

A1 --> "Short activity <img:sourceforge.jpg>"
@enduml

```



4.8 注释

你可以在活动定义之后用 `note left`, `note right`, `note top` or `note bottom`, 命令给活动添加注释。

如果想给开始点添加注释，只需把注释的定义放在活动图最开始的地方即可。

也可以用关键字 `endnote` 定义多行注释。

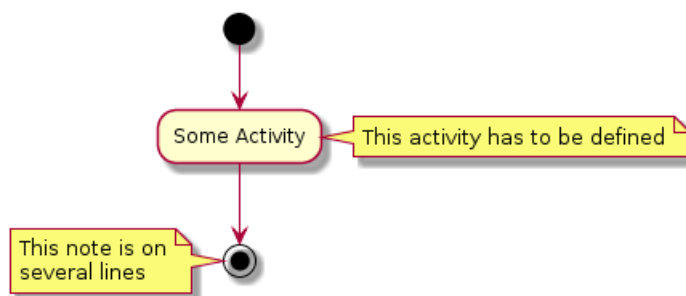
```

@startuml

(*) --> "Some Activity"
note right: This activity has to be defined
"Some Activity" --> (*)
note left
This note is on
several lines
end note

@enduml

```



4.9 分区

用关键字 `partition` 定义分区，还可以设置背景色 (用颜色名或者颜色值)。

定义活动的时候，它自动被放置到最新的分区中。
用}结束分区的定义。

```
@startuml

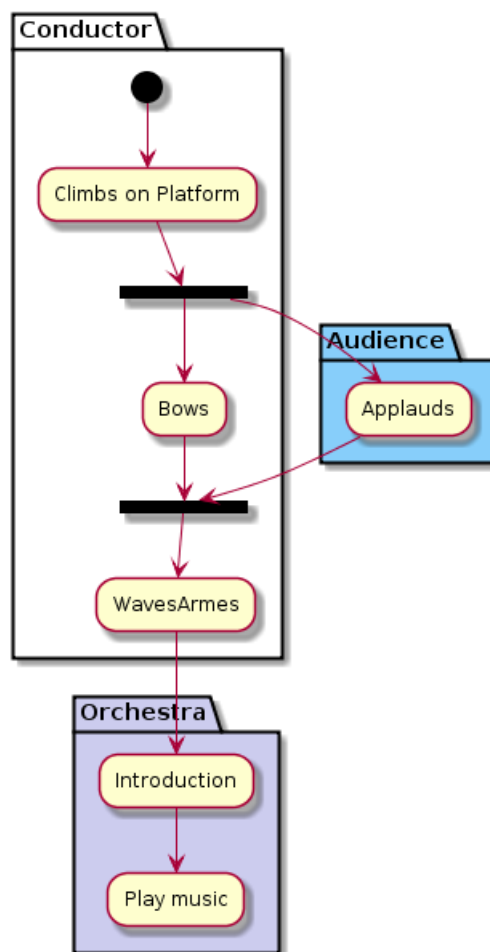
partition Conductor {
  (*) --> "Climbs on Platform"
  --> == S1 ==
  --> Bows
}

partition Audience LightSkyBlue {
  == S1 == --> Applauds
}

partition Conductor {
  Bows --> == S2 ==
  --> WavesArmes
  Applauds --> == S2 ==
}

partition Orchestra #CCCCEE {
  WavesArmes --> Introduction
  --> "Play music"
}

@enduml
```



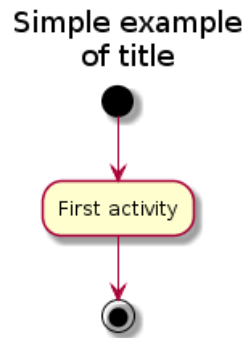
4.10 给图示添加标题

用 **title** 关键字给图示添加标题。

跟序列图中一样，你也可以用 `title` 和 `end title` 定义多行标题。

```
@startuml
title Simple example\nof title

(*) --> "First activity"
--> (*)
@enduml
```



4.11 显示参数

用 `skinparam` 命令修改字体和颜色。

如下场景可用：

- 在图示定义中
- 在引入的文件中
- 在命令行或 ANT 任务提供的配置文件中。

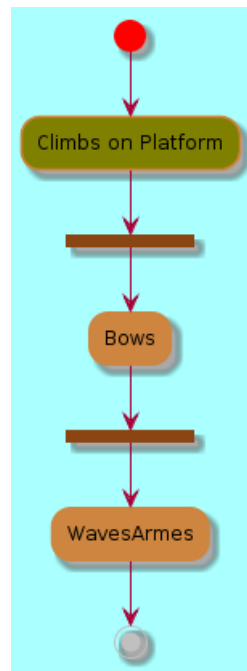
还可以为构造类型指定特殊颜色和字体。

```
@startuml

skinparam backgroundColor #AAFFFF
skinparam activity {
  StartColor red
  BarColor SaddleBrown
  EndColor Silver
  BackgroundColor Peru
  BackgroundColor<< Begin >> Olive
  BorderColor Peru
  FontName Impact
}

(*) --> "Climbs on Platform" << Begin >>
--> === S1 ===
--> Bows
--> === S2 ===
--> WavesArmes
--> (*)

@enduml
```



4.12 八边形活动

可用用 `skinparam activityShape octagon` 命令将活动的外形改为八边形。

```

@startuml
'Default is skinparam activityShape roundBox
skinparam activityShape octagon

(*) --> "First Activity"
"First Activity" --> (*)

@enduml
  
```



4.13 一个完整的例子

```

@startuml
title Servlet Container

(*) --> "ClickServlet.handleRequest()"
--> "new Page"

if "Page.onSecurityCheck" then
->[true] "Page.onInit()"

if "isForward?" then
->[no] "Process controls"

if "continue processing?" then
->[yes] ===RENDERING===
else
  
```

```

-->[no] ===REDIRECT_CHECK===
endif

else
-->[yes] ===RENDERING===
endif

if "is Post?" then
-->[yes] "Page.onPost()"
--> "Page.onRender()" as render
--> ===REDIRECT_CHECK===
else
-->[no] "Page.onGet()"
--> render
endif

else
-->[false] ===REDIRECT_CHECK===
endif

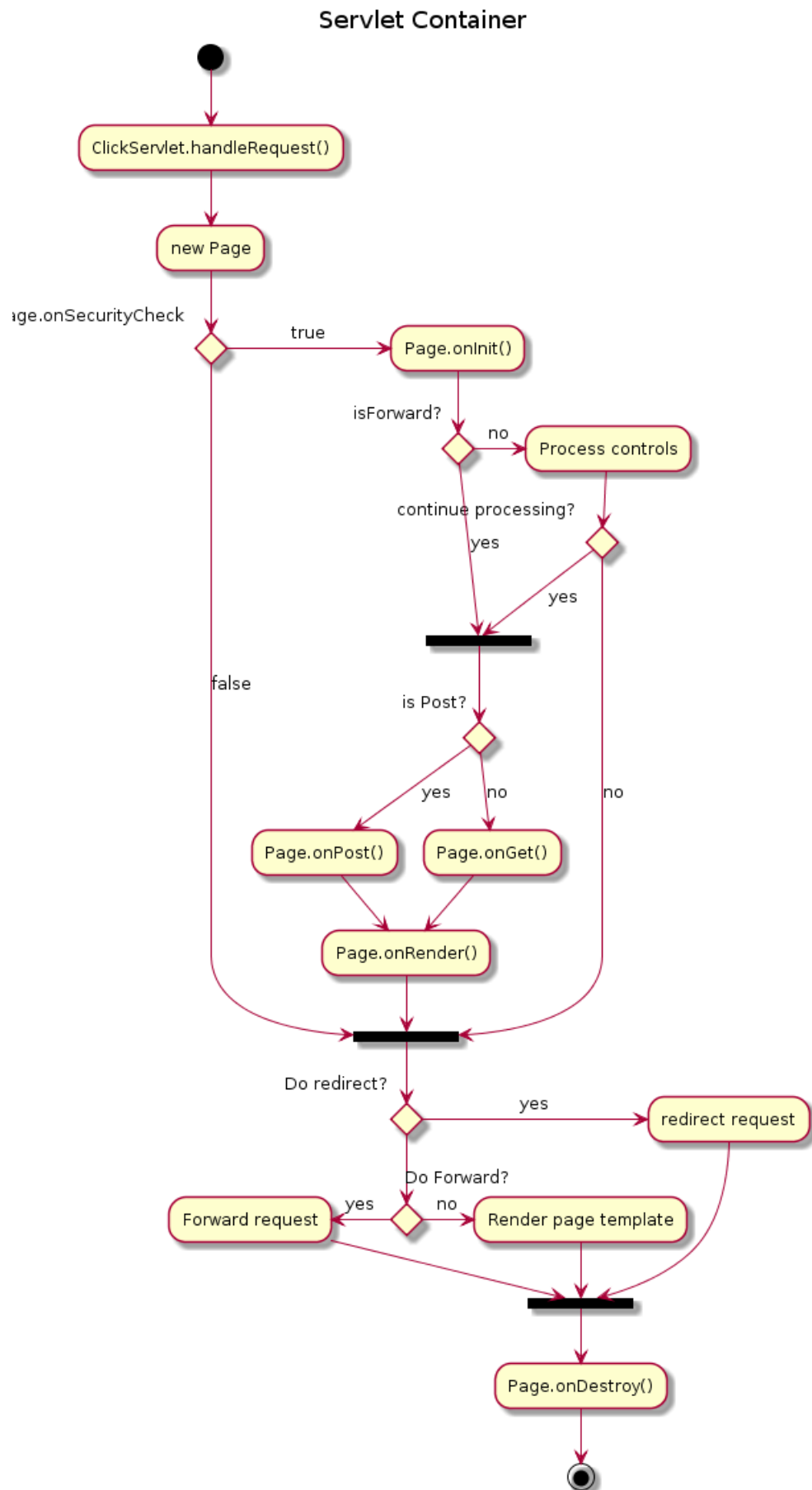
if "Do redirect?" then
->[yes] "redirect request"
--> ==BEFORE_DESTROY==
else
if "Do Forward?" then
-left->[yes] "Forward request"
--> ==BEFORE_DESTROY==
else
-right->[no] "Render page template"
--> ==BEFORE_DESTROY==
endif
endif

--> "Page.onDestroy()"
-->(*)

@enduml

```





5 活动图 (新语法)

当前活动图 (activity diagram) 的语法有诸多限制和缺点，比如代码难以维护。

所以从 V7947 开始提出一种全新的、更好的语法格式和软件实现供用户使用 (beta 版)。

就像序列图一样，新的软件实现的另一个优点是它不再依赖与 Graphviz。

新的语法将会替换旧的语法。然而考虑到兼容性，旧的语法仍被能够使用以确保向前兼容。

但是我们鼓励用户使用新的语法格式。

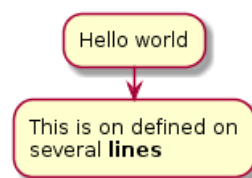
5.1 简单活动图

活动标签 (activity label) 以冒号开始，以分号结束。

文本格式支持 creole wiki 语法。

活动默认安装它们定义的顺序就行连接。

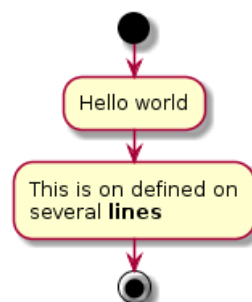
```
@startuml
:Hello world;
:This is on defined on
several **lines**;
@enduml
```



5.2 开始/结束

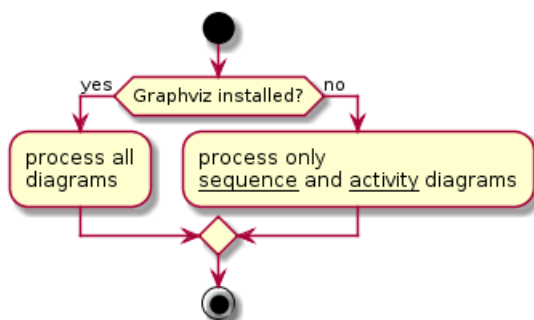
你可以使用关键字 **start** 和 **stop** 表示图示的开始和结束。

```
@startuml
start
:Hello world;
:This is on defined on
several **lines**;
stop
@enduml
```



You can also use the **end** keyword.

```
@startuml
start
:Hello world;
:This is on defined on
several **lines**;
end
@enduml
```

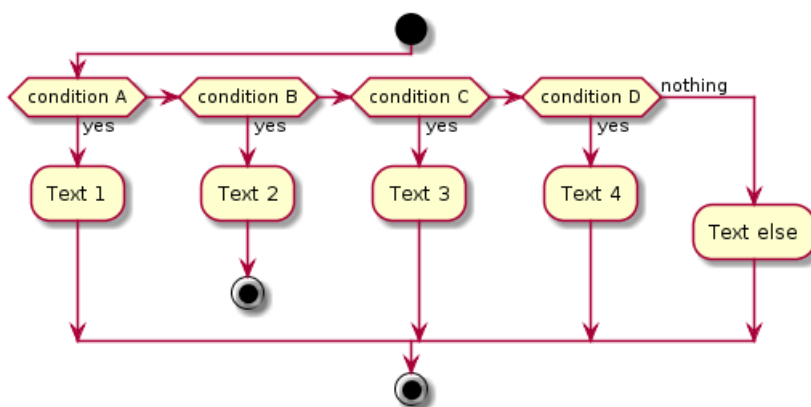


5.3 条件语句

在图示中可以使用关键字 `if`, `then` 和 `else` 设置分支测试。标注文字则放在括号中。

```

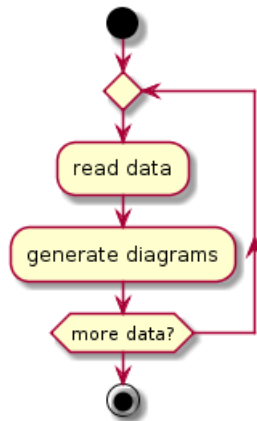
@startuml
start
if (Graphviz installed?) then (yes)
:process all\ndiagrams;
else (no)
:process only
__sequence__ and __activity__ diagrams;
endif
stop
@enduml
  
```



也可以使用关键字 `elseif` 设置多个分支测试。

```

@startuml
start
if (condition A) then (yes)
:Text 1;
elseif (condition B) then (yes)
:Text 2;
stop
elseif (condition C) then (yes)
:Text 3;
elseif (condition D) then (yes)
:Text 4;
else (nothing)
:Text else;
endif
stop
@enduml
  
```

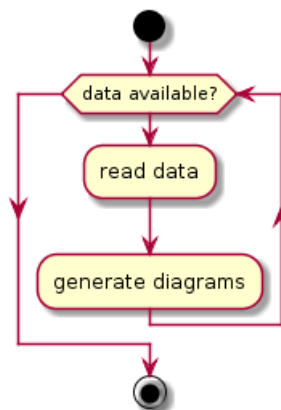


5.4 重复循环

你可以使用关键字 `repeat` 和 `repeatwhile` 进行重复循环。

```

@startuml
start
repeat
:read data;
:generate diagrams;
repeat while (more data?)
stop
@enduml
  
```

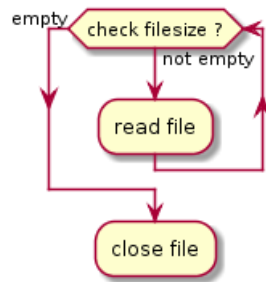


5.5 while 循环

可以使用关键字 `while` 和 `end while` 进行 while 循环。

```

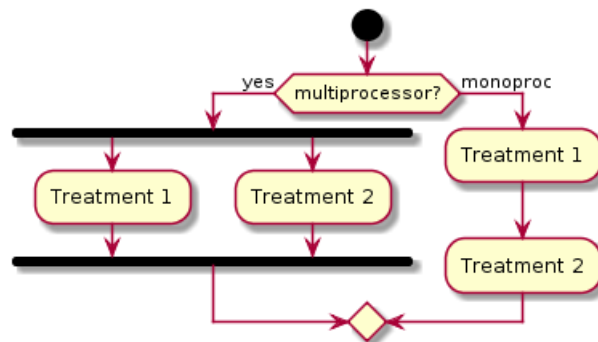
@startuml
start
while (data available?)
:read data;
:generate diagrams;
endwhile
stop
@enduml
  
```

还可以在关键字 `endwhile` 后添加标注，还有一种方式是使用关键字 `is`。

```

@startuml
while (check filesize ?) is (not empty)
:read file;
endwhile (empty)
:close file;
@enduml
  
```



5.6 并行处理

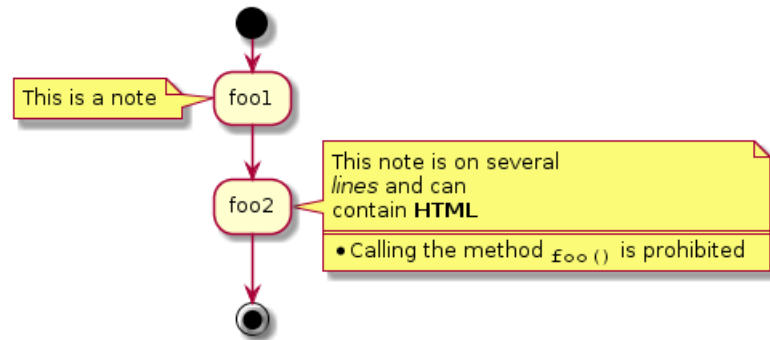
你可以使用关键字 `fork`, `fork again` 和 `end fork` 表示并行处理。

```

@startuml
start

if (multiprocessor?) then (yes)
fork
:Treatment 1;
fork again
:Treatment 2;
end fork
else (monoprocc)
:Treatment 1;
:Treatment 2;
endif

@enduml
  
```



5.7 注释

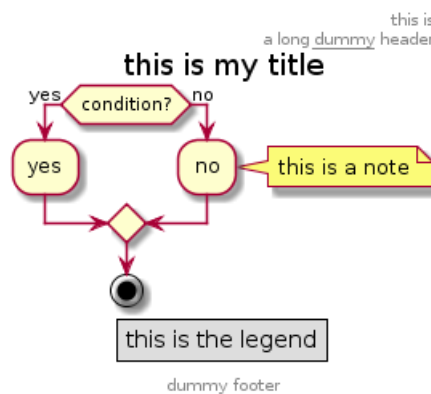
文本格式支持 creole wiki 语法。

```

@startuml

start
:foo1;
note left: This is a note
:foo2;
note right
This note is on several
//lines// and can
contain <b>HTML</b>
====
* Calling the method "foo()" is prohibited
end note
stop

@enduml
  
```



5.8 标题和图例

你可以给图表 (diagram) 添加标题、标头、脚注和图例。

```

@startuml
title this is my title
if (condition?) then (yes)
:yes;
else (no)
:no;
note right
this is a note
end note
endif
stop
  
```

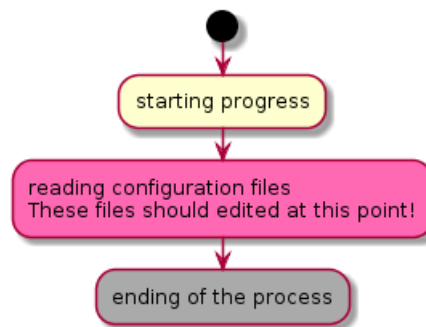
```

legend
this is the legend
endlegend

footer dummy footer
header
this is
a long __dummy__ header
end header

@enduml

```



5.9 颜色

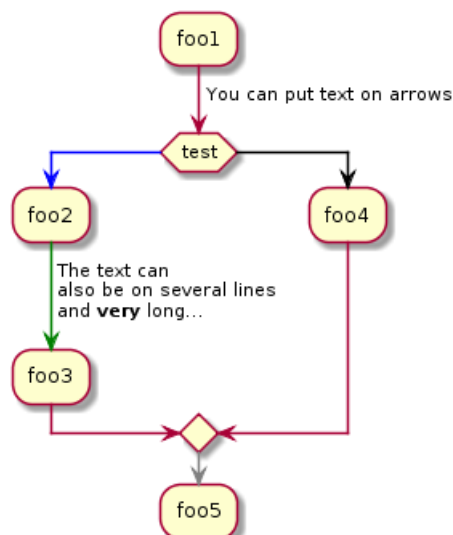
你可以为活动 (activity) 指定一种颜色。

```

@startuml
start
:starting progress;
#HotPink:reading configuration files
These files should edited at this point!;
#AAAAAA:ending of the process;

@enduml

```



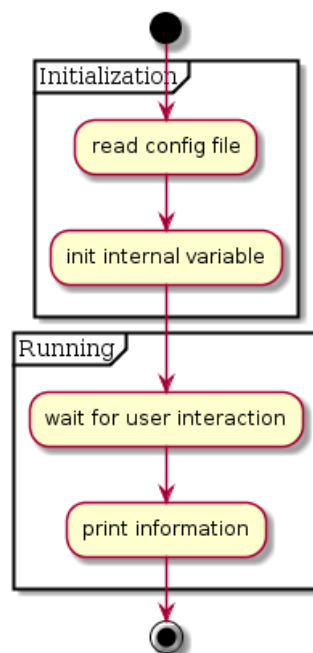
5.10 箭头

使用 -> 标记, 你可以给箭头添加文字或者修改箭头颜色。

```

@startuml
:foo1;
-> You can put text on arrows;
if (test) then
-[#blue]->
:foo2;
-[#green]-> The text can
also be on several lines
and **very** long...;
:foo3;
else
-[#black]->
:foo4;
endif
-[#gray]->
:foo5;
@enduml

```



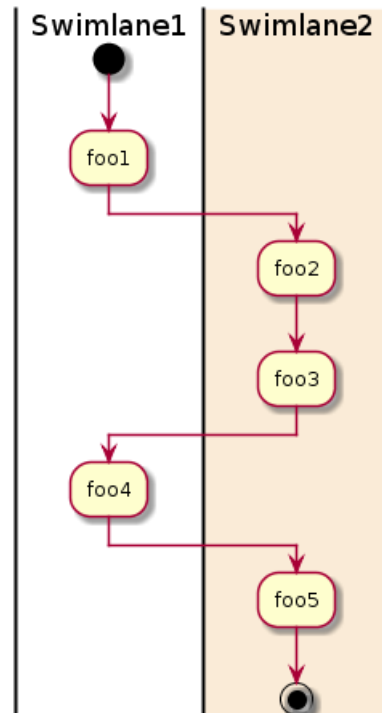
5.11 组合 (grouping)

通过定义分区 (partition)，你可以把多个活动组合 (group) 在一起。

```

@startuml
start
partition Initialization {
:read config file;
:init internal variable;
}
partition Running {
:wait for user interaction;
:print information;
}
stop
@enduml

```

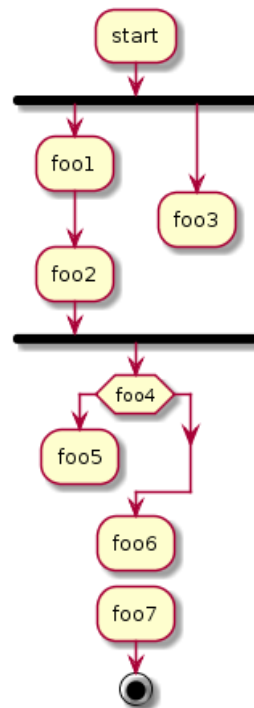


5.12 泳道 (Swimlanes)

你可以使用管道符 | 来定义泳道。

还可以改变泳道的颜色。

```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



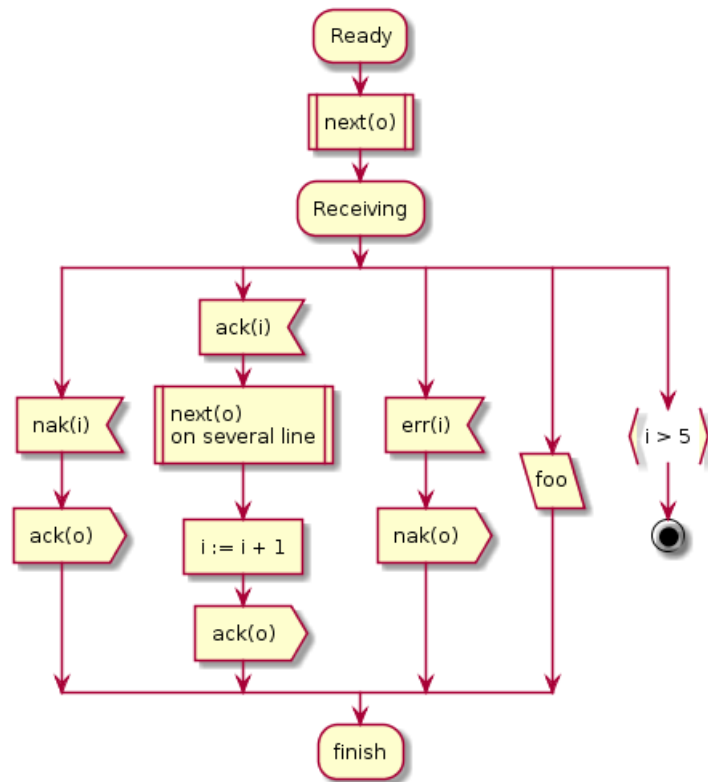
5.13 分离 (detach)

可以使用关键字 `detach` 移除箭头。

```

@startuml
: start;
fork
: foo1;
: foo2;
fork again
: foo3;
detach
endfork
if (foo4) then
: foo5;
detach
endif
: foo6;
detach
: foo7;
stop
@enduml

```



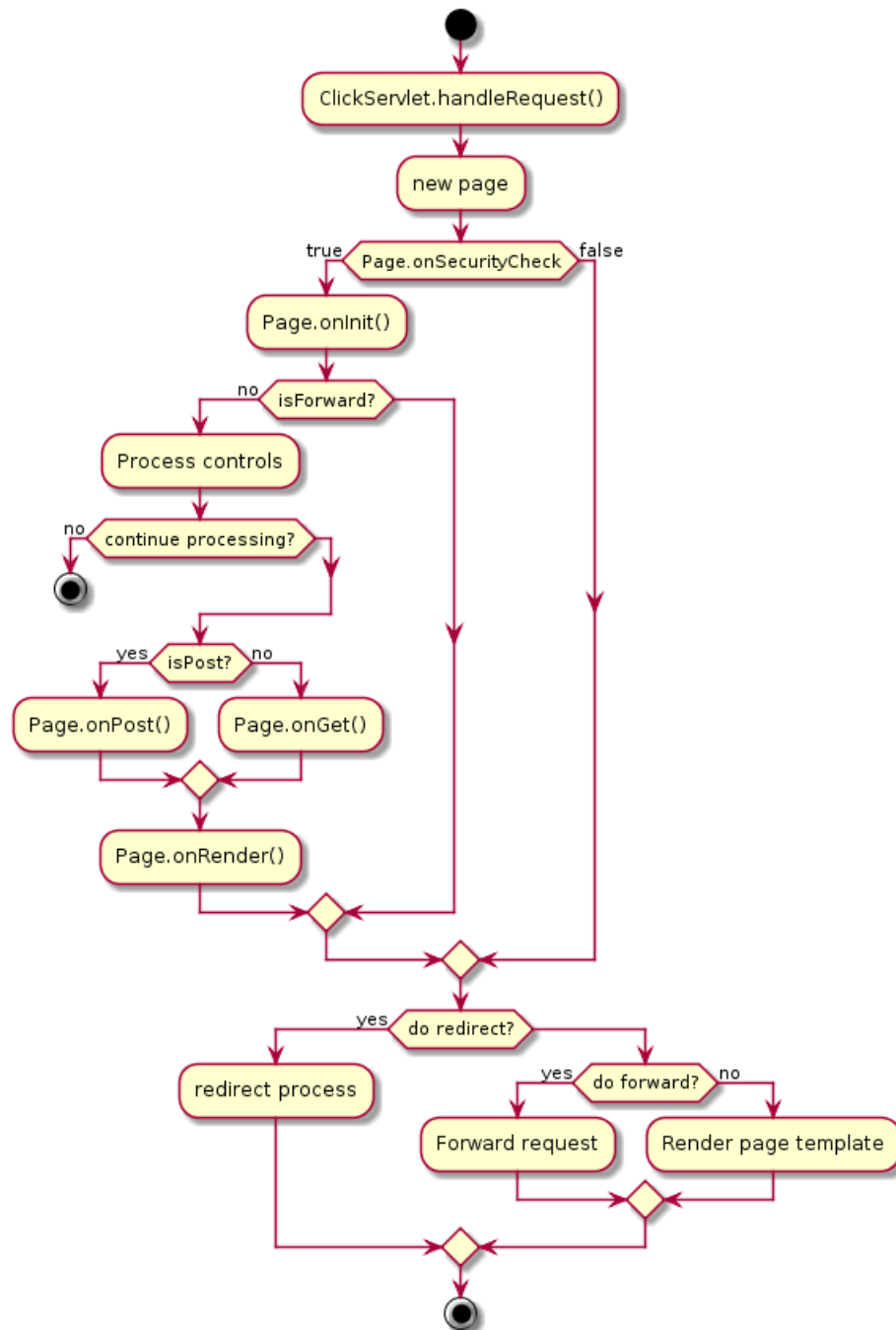
5.14 特殊领域语言 (SDL)

通过修改活动标签最后的分号分隔符 (;), 可以为活动设置不同的形状。

- |
- <
- >
- /
-]
- }

```

@startuml
:Ready;
:next(o)|
:Receiving;
split
:nak(i)<
:ack(o)>
split again
:ack(i)<
:next(o)
on several line|
:i := i + 1]
:ack(o)>
split again
:err(i)<
:nak(o)>
split again
:foo/
split again
:i > 5}
stop
end split
:finish;
@enduml
  
```



5.15 一个完整的例子

@startuml

```

start
:ClickServlet.handleRequest();
:new page;
if (Page.onSecurityCheck) then (true)
:Page.onInit();
if (isForward?) then (no)
:Process controls;
if (continue processing?) then (no)
stop
endif

```

```

if (isPost?) then (yes)
:Page.onPost();

```



```
else (no)
:Page.onGet();
endif
:Page.onRender();
endif
else (false)
endif

if (do redirect?) then (yes)
:redirect process;
else
if (do forward?) then (yes)
:Forward request;
else (no)
:Render page template;
endif
endif

stop

@enduml
```

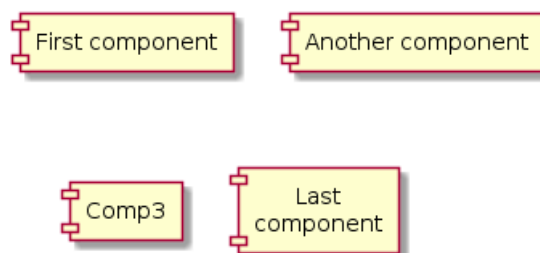
6 组件图

6.1 组件

组件必须用中括号括起来。

还可以使用关键字 `component` 定义一个组件。并且可以用关键字 `as` 给组件定义一个别名。这个别名可以在稍后定义关系的时候使用。

```
@startuml
[First component]
[Another component] as Comp2
component Comp3
component [Last\ncomponent] as Comp4
@enduml
```



6.2 接口

接口可以使用 `()` 来定义 (因为这个看起来像个圆)。

还可以使用关键字 `interface` 关键字来定义接口。并且还可以使用关键字 `as` 定义一个别名。这个别名可以在稍后定义关系的时候使用。

我们稍后可以看到，接口的定义是可选的。

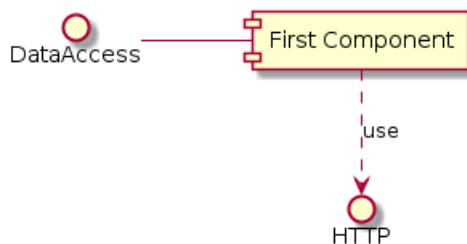
```
@startuml
() "First Interface"
() "Another interface" as Interf2
interface Interf3
interface "Last\ninterface" as Interf4
@enduml
```



6.3 基础的示例

元素之间可以使用虚线 (`..`)、直线 (`--`)、箭头 (`-->`) 进行连接。

```
@startuml
DataAccess - [First Component]
[First Component] ..> HTTP : use
@enduml
```



6.4 使用注释

你可以使用 `note left of`, `note right of`, `note top of`, `note bottom of` 等关键字定义相对于对象位置的注释。

也可以使用关键字 `note` 单独定义注释，然后使用虚线 (..) 将其连接到其他对象。

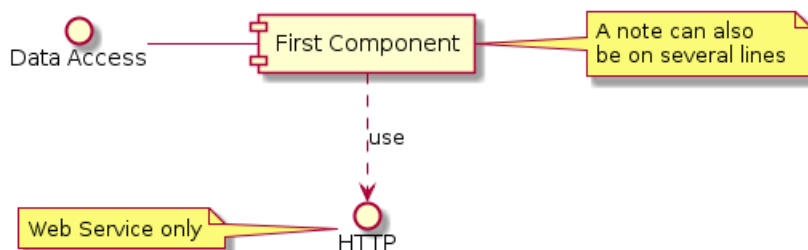
```
@startuml
interface "Data Access" as DA

DA - [First Component]
[First Component] ..> HTTP : use

note left of HTTP : Web Service only

note right of [First Component]
A note can also
be on several lines
end note

@enduml
```



6.5 组合组件

你可以使用多个关键字将组件和接口组合在一起。

- `package`
- `node`
- `folder`
- `frame`
- `cloud`
- `database`

```
@startuml
package "Some Group" {
HTTP - [First Component]
[Another Component]
}

node "Other Groups" {
FTP - [Second Component]
}
```

```

[First Component] --> FTP
}

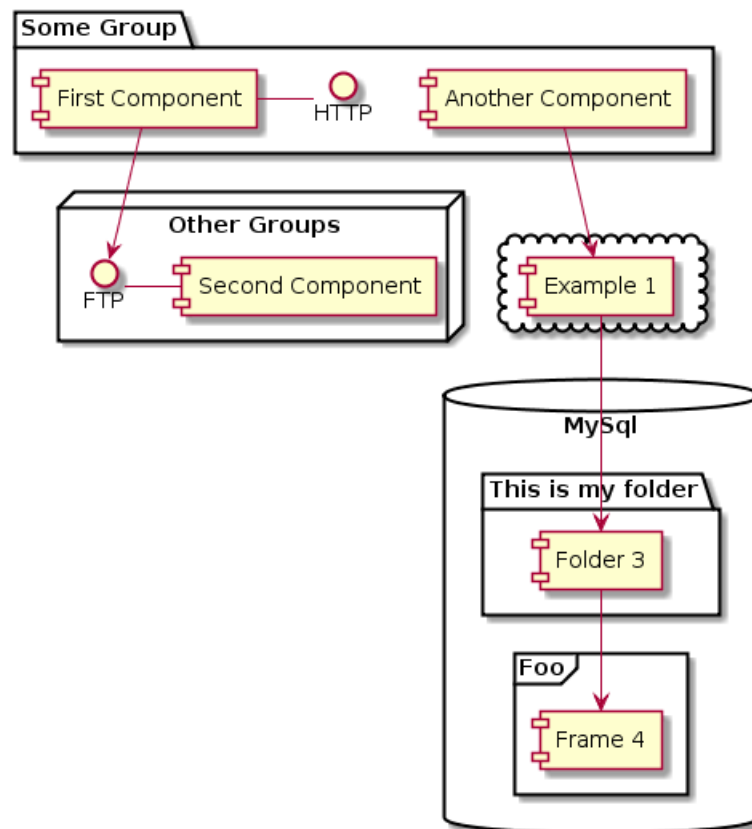
cloud {
[Example 1]
}

database "MySql" {
folder "This is my folder" {
[Folder 3]
}
frame "Foo" {
[Frame 4]
}
}

[Another Component] --> [Example 1]
[Example 1] --> [Folder 3]
[Folder 3] --> [Frame 4]

@enduml

```



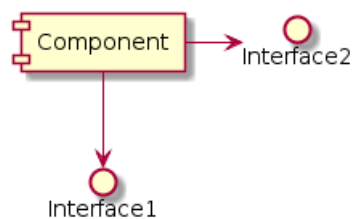
6.6 改变箭头方向

默认情况下，对象之间用 -- 连接，并且连接是竖直的。不过可以使用一个横线或者点设置水平方向的连接，就行这样：

```

@startuml
[Component] --> Interface1
[Component] -> Interface2
@enduml

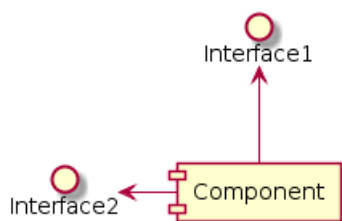
```



也可以使用反向连接:

```

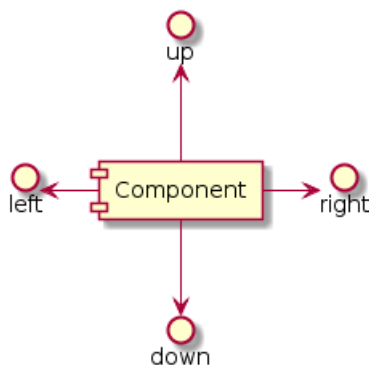
@startuml
Interface1 <-- [Component]
Interface2 <- [Component]
@enduml
  
```



还可以使用关键字 `left`, `right`, `up` or `down` 改变箭头方向。

```

@startuml
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@enduml
  
```



允许使用方向单词的首字母或者前两个字母表示方向 (例如 `-d-`, `-do-`, `-down-` 都是等价的)。
请不要乱用这些功能: *Graphviz*(PlantUML 的后端引擎) 不喜欢这个样子。

6.7 添加图示标题

关键字 `title` 用于添加标题。

你可以在序列图中使用关键字 `title` 和 `end title` 添加长标题。

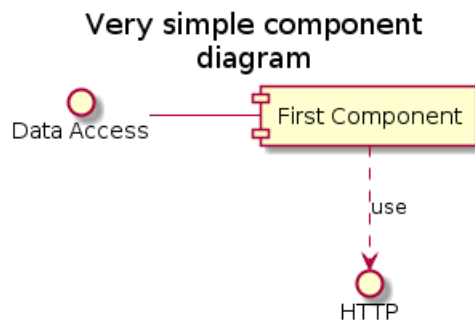
```

@startuml
title Very simple component\ndiagram

interface "Data Access" as DA

DA - [First Component]
[First Component] ..> HTTP : use

@enduml
  
```



6.8 使用 UML2 标记符

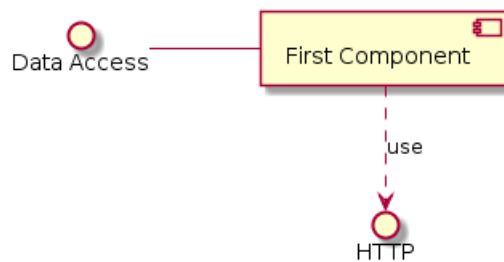
命令 `skinparam componentStyle uml2` 可以切换到 UML2 标记符。

```
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

DA - [First Component]
[First Component] ..> HTTP : use

@enduml
```



6.9 Individual colors

你可以在声明一个组件时加上颜色的声明。

```
@startuml
component [Web Server] #Yellow
@enduml
```



6.10 显示参数

可以使用命令 `skinparam` 改变字体和颜色。

你可以在如下场景使用这些命令：

- 在图示的定义中，
- 在包含进来的文件中，
- 在命令行或者 ANT 任务提供的配置文件中。

可以为构造类型和接口定义特殊的颜色和字体。

```

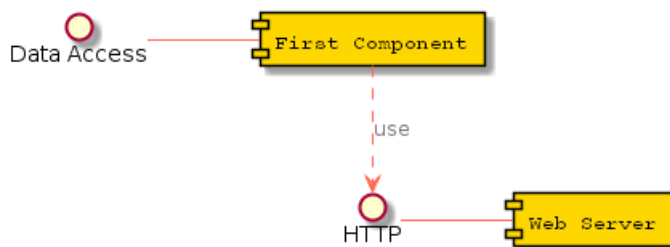
@startuml
skinparam component {
FontSize 13
InterfaceBackgroundColor RosyBrown
InterfaceBorderColor orange
BackgroundColor<<Apache>> Red
BorderColor<<Apache>> #FF6655
FontName Courier
BorderColor black
BackgroundColor gold
ArrowFontName Impact
ArrowColor #FF6655
ArrowFontColor #777777
}

() "Data Access" as DA

DA - [First Component]
[First Component] ..> () HTTP : use
HTTP - [Web Server] << Apache >>

@enduml

```



```

@startuml
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>

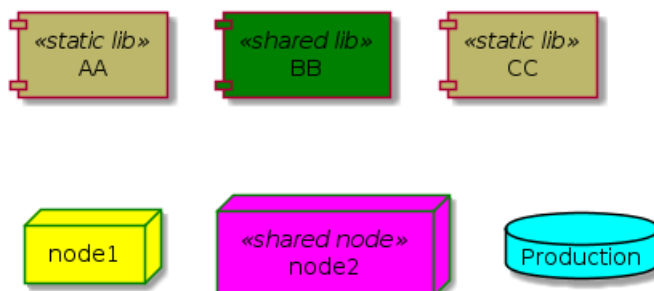
node node1
node node2 <<shared node>>
database Production

skinparam component {
backgroundColor<<static lib>> DarkKhaki
backgroundColor<<shared lib>> Green
}

skinparam node {
borderColor Green
backgroundColor Yellow
backgroundColor<<shared node>> Magenta
}
skinparam databaseBackgroundColor Aqua

@enduml

```



7 状态图

7.1 简单状态

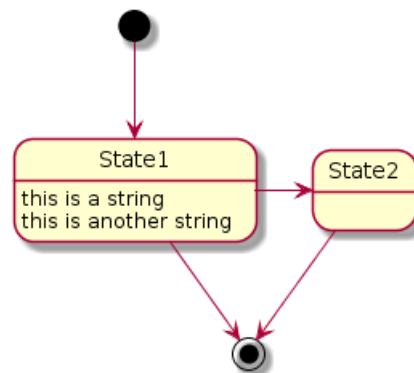
使用 ([*]) 开始和结束状态图。

使用 --> 添加箭头。

```
@startuml
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

State1 -> State2
State2 --> [*]

@enduml
```



7.2 合成状态

一个状态也可能是合成的，必须使用关键字 **state** 和花括号来定义合成状态。

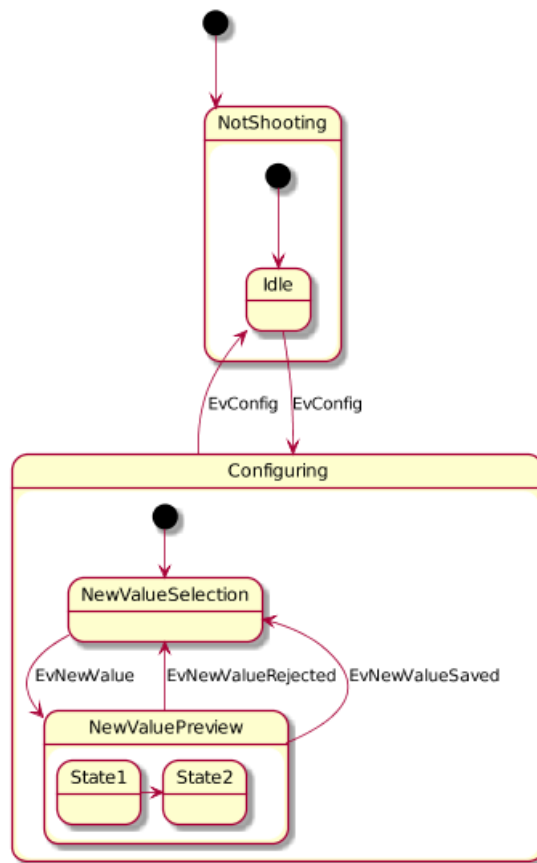
```
@startuml
scale 350 width
[*] --> NotShooting

state NotShooting {
[*] --> Idle
Idle --> Configuring : EvConfig
Configuring --> Idle : EvConfig
}

state Configuring {
[*] --> NewValueSelection
NewValueSelection --> NewValuePreview : EvNewValue
NewValuePreview --> NewValueSelection : EvNewValueRejected
NewValuePreview --> NewValueSelection : EvNewValueSaved
}

state NewValuePreview {
State1 -> State2
}

}
@enduml
```

7.3 长名字

也可以使用关键字 `state` 定义长名字状态。

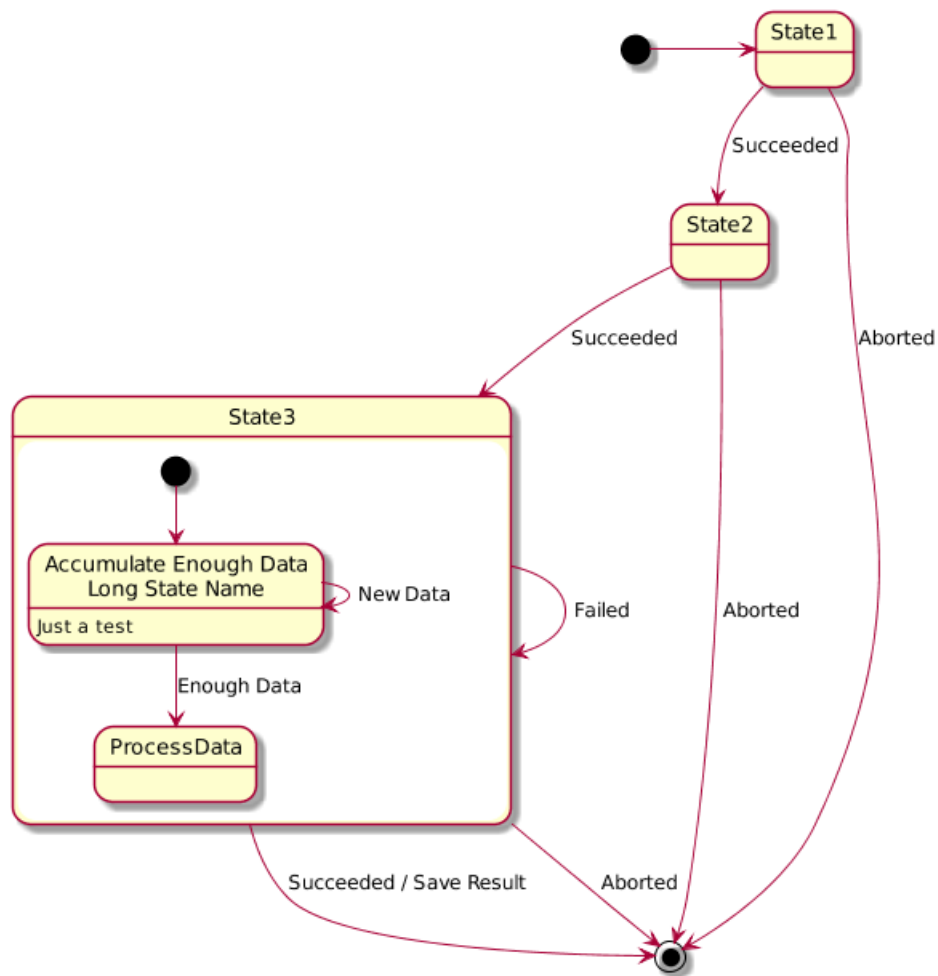
```

@startuml
scale 600 width

[*] -> State1
State1 --> State2 : Succeeded
State1 --> [*] : Aborted
State2 --> State3 : Succeeded
State2 --> [*] : Aborted
state State3 {
state "Accumulate Enough Data\nLong State Name" as long1
long1 : Just a test
[*] --> long1
long1 --> long1 : New Data
long1 --> ProcessData : Enough Data
}
State3 --> State3 : Failed
State3 --> [*] : Succeeded / Save Result
State3 --> [*] : Aborted

@enduml

```



7.4 并发状态

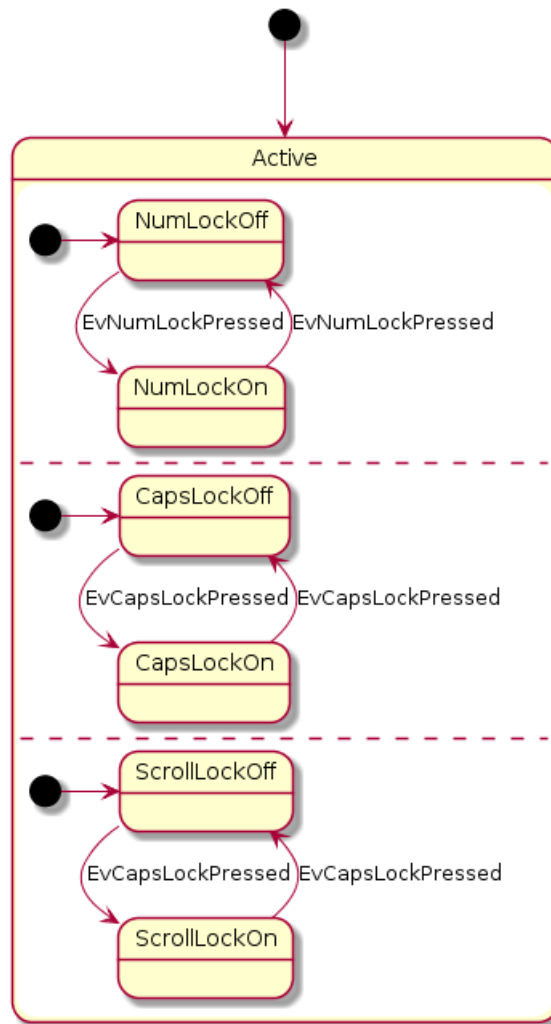
用 --作为分隔符来合成并发状态。

```

@startuml
[*] --> Active

state Active {
    [*] --> NumLockOff
    NumLockOff --> NumLockOn : EvNumLockPressed
    NumLockOn --> NumLockOff : EvNumLockPressed
    --
    [*] --> CapsLockOff
    CapsLockOff --> CapsLockOn : EvCapsLockPressed
    CapsLockOn --> CapsLockOff : EvCapsLockPressed
    --
    [*] --> ScrollLockOff
    ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
    ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}

@enduml
  
```



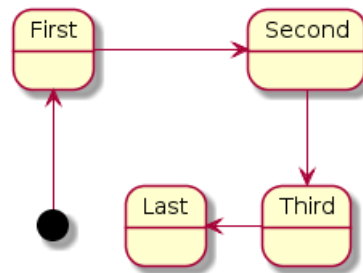
7.5 箭头方向

使用 `->` 定义水平箭头，也可以使用下列格式强制设置箭头方向：

- `-down->` (default arrow)
- `-right->` or `->`
- `-left->`
- `-up->`

```

@startuml
[*] -up-> First
First -right-> Second
Second --> Third
Third -left-> Last
@enduml
  
```



可以用首字母缩写或者开始的两个字母定义方向 (如, -d-, -down-和 -do-是完全等价的)。
请不要滥用这些功能, *Graphviz* 不喜欢这样。

7.6 注释

可以用 `note left of`, `note right of`, `note top of`, `note bottom of` 关键字来定义注释。
还可以定义多行注释。

```

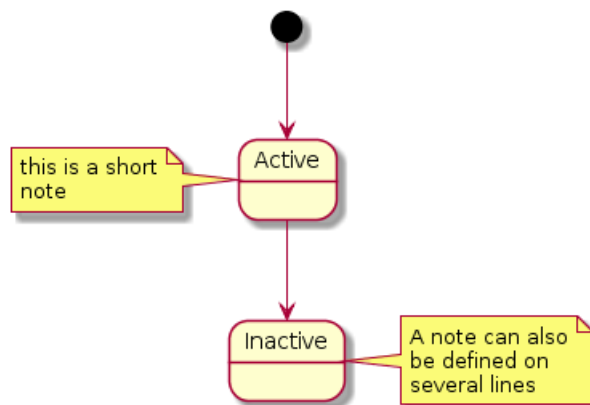
@startuml

[*] --> Active
Active --> Inactive

note left of Active : this is a short\nnote

note right of Inactive
A note can also
be defined on
several lines
end note

@enduml
  
```



以及浮动注释。

```

@startuml

state foo
note "This is a floating note" as N1

@enduml
  
```



7.7 更多注释

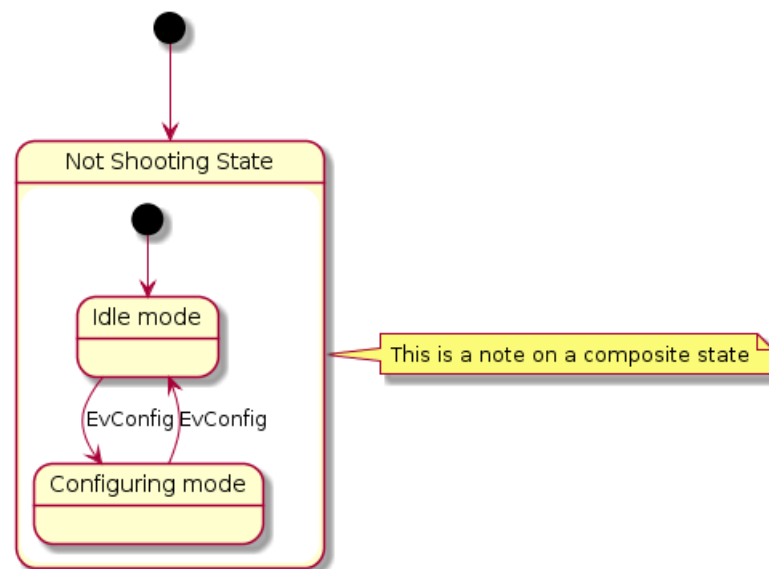
可以在合成状态中放置注释。

```
@startuml
[*] --> NotShooting

state "Not Shooting State" as NotShooting {
state "Idle mode" as Idle
state "Configuring mode" as Configuring
[*] --> Idle
Idle --> Configuring : EvConfig
Configuring --> Idle : EvConfig
}

note right of NotShooting : This is a note on a composite state

@enduml
```



7.8 显示参数

使用 `skinparam` 命令改变字体和颜色。

在如下场景使用：

- 在图示定义中，
- 在包含进来的文件中，
- 在命令行或 ANT 任务提供的配置文件中。

还可以为状态的构造类型指定特殊的字体和颜色。

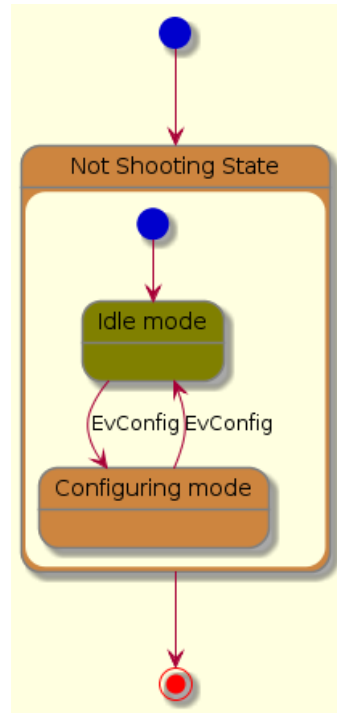
```
@startuml
skinparam backgroundColor LightYellow
skinparam state {
StartColor MediumBlue
EndColor Red
BackgroundColor Peru
BackgroundColor<<Warning>> Olive
BorderColor Gray
FontName Impact
}

[*] --> NotShooting

state "Not Shooting State" as NotShooting {
```

```
state "Idle mode" as Idle <<Warning>>
state "Configuring mode" as Configuring
[*] --> Idle
Idle --> Configuring : EvConfig
Configuring --> Idle : EvConfig
}

NotShooting --> [*]
@enduml
```

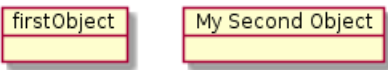


8 对象图

8.1 对象的定义

使用关键字 `object` 定义实例。

```
@startuml
object firstObject
object "My Second Object" as o2
@enduml
```



8.2 对象之间的关系

对象之间的关系用如下符号定义：

继承	< --	
合成	*--	
聚合	o--	

也可以用“..”来代替“--”以使用点线。

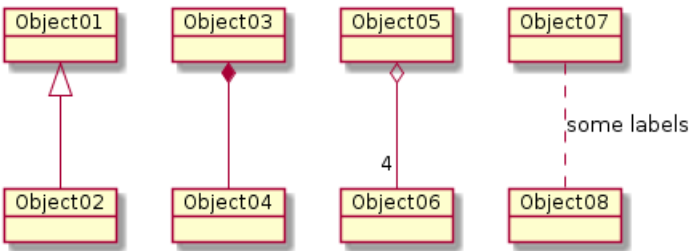
知道了这些规则，就可以画下面的图：

可以用冒号给关系添加标签，标签内容紧跟在冒号之后。

用双引号在关系的两边添加基数。

```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

Object01 <|-- Object02
Object03 *-- Object04
Object05 o-- "4" Object06
Object07 .. Object08 : some labels
@enduml
```

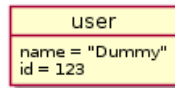


8.3 添加属性

用冒号加属性名的形式声明属性。

```
@startuml
object user

user : name = "Dummy"
user : id = 123
@enduml
```

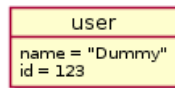


也可以用大括号批量声明属性。

```
@startuml
```

```
object user {
name = "Dummy"
id = 123
}
```

```
@enduml
```



8.4 类图中的通用特性

- 可见性
- 定义注释
- 使用包
- 添加标题
- 美化输出内容
- 分割图片

9 通用命令

9.1 脚注和标头

你可以使用 `header` 或 `footer` 给任何图示添加脚注或标头

还可以 (可选) 使用关键字 `center`, `left` 或 `right` 设置脚注或标头位置。

也可以定义多行脚注或标头。

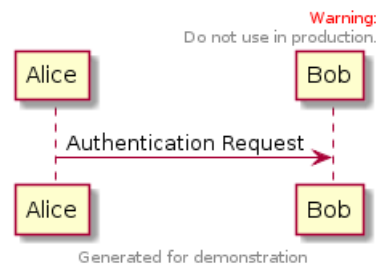
还可以在标头或脚注中放置 HTML 语句。

```
@startuml
Alice -> Bob: Authentication Request

header
<font color=red>Warning:</font>
Do not use in production.
endheader

center footer Generated for demonstration

@enduml
```



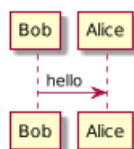
9.2 缩放

你可以用缩放命令来调整生成的图像

你可以指定缩放因子你还可以指定宽度或者高度（像素）你也可以同时指定宽度和高度：图像将被缩放到适合给定的大小。

- `scale 1.5`
- `scale 2/3`
- `scale 200 width`
- `scale 200 height`
- `scale 200*100`
- `scale max 300*200`

```
@startuml
scale 180*90
Bob->Alice : hello
@enduml
```



10 Salt

Salt is a subproject included in PlantUML that may help you to design graphical interface. You can use either `@startsalt` keyword, or `@startuml` followed by a line with `salt` keyword.

10.1 Basic widgets

A window must start and end with brackets. You can then define:

- Button using `[` and `]`.
- Radio button using `(` and `)`.
- Checkbox using `[` and `]`.
- User text area using `"`.

```
@startuml
salt
{
Just plain text
[This is my button]
() Unchecked radio
(X) Checked radio
[] Unchecked box
[X] Checked box
"Enter text here "
^This is a droplist^
}
@enduml
```

The goal of this tool is to discuss about simple and sample windows.

10.2 Using grid

A table is automatically created when you use an opening bracket `{`. And you have to use `|` to separate columns.

For example:

```
@startsalt
{
Login      | "MyName   "
Password   | "****     "
[Cancel]   | [ OK     ]
}
@endsalt
```

Just after the opening bracket, you can use a character to define if you want to draw lines or columns of the grid :

To display all vertical and horizontal lines

! To display all vertical lines

- To display all horizontal lines

+ To display external lines

```
@startsalt
{+
Login      | "MyName   "
Password   | "****     "
[Cancel]   | [ OK     ]
}
@endsalt
```



10.3 Using separator

You can use several horizontal lines as separator.

```
@startsalt
{
Text1
..
"Some field"
==
Note on usage
~~
Another text
--
[Ok]
}
@endsalt
```

10.4 Tree widget

To have a Tree, you have to start with {T and to use + to denote hierarchy.

```
@startsalt
{
{T
+ World
++ America
+++ Canada
+++ USA
++++ New York
++++ Boston
+++ Mexico
++ Europe
+++ Italy
+++ Germany
++++ Berlin
++ Africa
}
}
@endsalt
```

10.5 Enclosing brackets

You can define subelements by opening a new opening bracket.

```
@startsalt
{
Name          | "          "
Modifiers:    | { (X) public | () default | () private | () protected
[] abstract | [] final    | [] static }
Superclass:   | { "java.lang.Object " | [Browse...] }
}
@endsalt
```

10.6 Adding tabs

You can add tabs using {/ notation. Note that you can use HTML code to have bold text.



```

@startsalt
{+
{/ <b>General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt

```

Tab could also be vertically oriented:

```

@startsalt
{+
{/ <b>General
Fullscreen
Behavior
Saving } |
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
[Close]
}
}
@endsalt

```

10.7 Using menu

You can add a menu by using {* notation.

```

@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt

```

It is also possible to open a menu:

```

@startsalt
{+
{* File | Edit | Source | Refactor
Refactor | New | Open File | - | Close | Close All }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt

```



10.8 Advanced table

You can use two special notations for table :

- * to indicate that a cell with span with left
- . to denotate an empty cell

```
@startsalt
{#
. | Column 2 | Column 3
Row header 1 | value 1 | value 2
Row header 2 | A long cell | *
}
@endsalt
```

11 Creole

A light Creole engine have been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

(Note that ascending compatibility with HTML syntax is preserved)

11.1 Emphasized text

```
@startuml
Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
This is bold
This is italics
This is "monospaced"
This is --stroked--
This is __underlined__
This is ~~waved~~
end note
@enduml
```

11.2 List

```
@startuml
object demo {
* Bullet list
* Second item
** Sub item
}

legend
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
end legend
@enduml
```

11.3 Escape character

You can use the tilde ~ to escape special creole characters.

```
@startuml
object demo {
This is not ~___underscored___.
This is not ~"monospaced".
}
@enduml
```



11.4 Horizontal lines

```
@startuml
database DB1 as "
You can have horizontal line
----
Or double line
====
Or strong line
----
Or dotted line
..My title..
Enjoy!
"
note right
This is working also in notes
You can also add title in all these lines
==Title==
--Another title--
end note

@enduml
```

11.5 Headings

```
@startuml
usecase UC1 as "
= Extra-large heading
Some text
== Large heading
Other text
=== Medium heading
Information
....
==== Small heading"
@enduml
```

11.6 Legacy HTML

Some HTML tags are also working:

- `` for bold text
- `<u>` or `<u:#AAAAAA>` or `<u:colorName>` for underline
- `<i>` for italic
- `<s>` or `<s:#AAAAAA>` or `<s:colorName>` for strike text
- `<w>` or `<w:#AAAAAA>` or `<w:colorName>` for wave underline text
- `<color:#AAAAAA>` or `<color:colorName>`
- `<back:#AAAAAA>` or `<back:colorName>` for background color
- `<size:nn>` to change font size
- `<img:file>` : the file must be accessible by the filesystem
- `<img:http://url>` : the URL must be available from the Internet

```
@startuml
:* You can change <color:red>text color</color>
* You can change <back:cadetblue>background color</back>
* You can change <size:18>size</size>
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
```



```

----
* Use image : <img:sourceforge.jpg>
;

@enduml

```

11.7 Table

```

@startuml
skinparam titleFontSize 14
title
Example of simple table
|= |= table |= header |
| a | table | row |
| b | table | row |
end title
[*] --> State1
@enduml

```

11.8 Tree

You can use |_ characters to build a tree.

```

@startuml
skinparam titleFontSize 14
title
Example of Tree
|_ First line
|_ **Bom(Model)**
|_ prop1
|_ prop2
|_ prop3
|_ Last line
end title
[*] --> State1
@enduml

```

11.9 Special characters

It's possible to use any unicode characters with &# syntax or <U+XXXX>

```

@startuml
usecase foo as "this is &#8734; long"
usecase bar as "this is also <U+221E> long"
@enduml

```

11.10 OpenIconic

OpenIconic is an very nice open source icon set. Those icons have been integrated into the creole parser, so you can use them out-of-the-box.

You can use the following syntax: <&ICON_NAME>.




```
@startuml
title: <size:20><&heart>Use of OpenIconic<&heart></size>
class Wifi
note left
Click on <&wifi>
end note
@enduml
```

The complete list is available on OpenIconic Website, or you can use the following special diagram:

```
@startuml
listopeniconic
@enduml
```



11.11 Defining and using sprites

A *Sprite* is a small graphic element that can be used in diagrams.

In PlantUML, sprites are monochrome and can have either 4, 8 or 16 gray level.

To define a sprite, you have to use a hexadecimal digit between 0 and F per pixel.

Then you can use the sprite using <\$XXX> where XXX is the name of the sprite.

```
@startuml
sprite $foo1 {
FFFFFFFFFFFFFFFF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
F0123456789ABCF
FFFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1>
@enduml
```

11.12 Encoding Sprite

To encode sprite, you can use the command line like:

```
java -jar plantuml.jar -encodesprite 16z foo.png
```

where `foo.png` if the image file you want to use (it will be converted to gray automatically).

After `-encodesprite`, you have to specify a format: 4, 8, 16, 4z, 8z or 16z.

The number indicates the gray level and the optionnal `z` is used to enable compression in sprite definition.

11.13 Importing Sprite

You can also launch the GUI to generate a sprite from an existing image.

Click in the menubar then on **File/Open Sprite Window**.

After copying an image into you clipboard, several possible definitions of the corresponding sprite will be displayed : you will just have to pickup the one you want.

11.14 Examples

```
@startuml
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj7lHWpa1XC716szOPq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPE
start
:click on <$printer> to print the page;
@enduml
```



```

@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p_FEjQEqB1z27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MHM-BLRyywPhrrlw3qu
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPE
sprite $disk {
444445566677881
436000000009991
43600000000ACA1
53700000001A7A1
537000000012B8A1
5380000000123B8A1
638000001233C9A1
634999AABBC99B1
744566778899AB1
7456AAAAA99AAB1
8566AFC228AABB1
8567AC8118BBBB1
867BD4433BBBBB1
39AAAAABBBBBBC1
}

title Use of sprites (<$printer>, <$bug>...)

class Example {
Can have some bug : <$bug>
Click on <$disk> to save
}

note left : The printer <$printer> is available

@enduml

```

12 修改字体和颜色

12.1 使用方法

使用 `skinparam` 命令修改字体和颜色。例如:

```
skinparam backgroundColor yellow
```

可以在如下场景中使用:

- 在图示的定义中,
- 在包含进来的文件中 (见预处理)。
- 在命令行或 ANT 任务提供的配置文件中。

12.2 嵌套

为了避免重复, 可以使用嵌套的定义。所以下面的定义:

```
skinparam xxxxParam1 value1
skinparam xxxxParam2 value2
skinparam xxxxParam3 value3
skinparam xxxxParam4 value4
```

完全等价于:

```
skinparam xxxx {
    Param1 value1
    Param2 value2
    Param3 value3
    Param4 value4
}
```

12.3 颜色

既可以使用颜色的标准名字也可以使用 RGB 颜色值。

参数名字	默认值	颜色	注释
backgroundColor	white		页面背景色
activityArrowColor	#A80036		活动图中箭头的颜色
activityBackgroundColor	#FEFCE		活动的背景色
activityBorderColor	#A80036		活动边界的颜色
activityStartColor	black		活动图开始的圆圈的颜色
activityEndColor	black		活动图结束的圆圈的颜色
activityBarColor	black		活动图中同步条的颜色
usecaseArrowColor	#A80036		usecase 图中箭头的颜色
usecaseActorBackgroundColor	#FEFCE		usecase 图中角色头部的颜色
usecaseActorBorderColor	#A80036		usecase 图中角色的边界的颜色
usecaseBackgroundColor	#FEFCE		usecase 的背景色
usecaseBorderColor	#A80036		usecase 图中 usecase 边界的颜色
classArrowColor	#A80036		类图中箭头的颜色
classBackgroundColor	#FEFCE		类图中类、接口、枚举的背景色
classBorderColor	#A80036		类图中类、接口、枚举的边界颜色
packageBackgroundColor	#FEFCE		类图中包的背景色
packageBorderColor	#A80036		类图中包的边界颜色
stereotypeCBackgroundColor	#ADD1B2		类图中圆圈的背景色
stereotypeABackgroundColor	#A9DCDF		类图中抽象类圆圈的背景色
stereotypeIBackgroundColor	#B4A7E5		类图中接口圆圈的背景色
stereotypeEBackgroundColor	#EB937F		类图中枚举圆圈的背景色
componentArrowColor	#A80036		组件图中箭头的颜色
componentBackgroundColor	#FEFCE		组件的背景色
componentBorderColor	#A80036		组件的边界颜色
componentInterfaceBackgroundColor	#FEFCE		组件图中接口的背景色
componentInterfaceBorderColor	#A80036		组件图中接口的边界颜色
noteBackgroundColor	#FBFB77		注释的背景色
noteBorderColor	#A80036		注释的边界颜色
stateBackgroundColor	#FEFCE		状态图中状态的背景色
stateBorderColor	#A80036		状态图中状态的边界颜色
stateArrowColor	#A80036		状态图中箭头的颜色
stateStartColor	black		状态图中开始圆圈的颜色
stateEndColor	black		状态图中结束圆圈的颜色
sequenceArrowColor	#A80036		序列图中箭头的颜色
sequenceActorBackgroundColor	#FEFCE		序列图中角色的头部颜色
sequenceActorBorderColor	#A80036		序列图中角色的边界颜色
sequenceGroupBackgroundColor	#EEEEEE		序列图中 alt/opt/loop 的标头颜色
sequenceLifeLineBackgroundColor	white		序列图中生命线的背景色
sequenceLifeLineBorderColor	#A80036		序列图中生命线的边界的颜色
sequenceParticipantBackgroundColor	#FEFCE		序列图中参与者的背景色
sequenceParticipantBorderColor	#A80036		序列图中参与者的边界颜色

12.4 字体的颜色、名字和尺寸

用如下参数改变字体属性: `xxxFontColor` `xxxFontSize` and `xxxFontName`。

例如:

```
skinparam classFontColor red
skinparam classFontSize 10
skinparam classFontName Apex
```

还可以改变默认的字体属性, 像这样: `skinparam defaultFontName`。

例如:

```
skinparam defaultFontName Apex
```

注意字体名称有很高的系统依赖性, 如果想有好的移植性, 就不要过度依赖于字体名字。移植性

参数 名字	默认 值	注释
activityFontColor activityFontSize activityFontStyle activityFontName	black 14 plain	用于活动框
activityArrowFontColor activityArrowFontSize activityArrowFontStyle activityArrowFontName	black 13 plain	用于活动图中的箭头之上的文字
circledCharacterFontColor circledCharacterFontSize circledCharacterFontStyle circledCharacterFontName circledCharacterRadius	black 17 bold Courier 11	用于类、枚举等元素的圆圈中的文字
classArrowFontColor classArrowFontSize classArrowFontStyle classArrowFontName	black 10 plain	用于类图中的箭头之上的文字
classAttributeFontColor classAttributeFontSize classAttributeIconSize classAttributeFontStyle classAttributeFontName	black 10 10 plain	类的属性和方法
classFontColor classFontSize classFontStyle classFontName	black 12 plain	用于类名
classStereotypeFontColor classStereotypeFontSize classStereotypeFontStyle classStereotypeFontName	black 12 italic	用于类中的构造类型
componentFontColor componentFontSize componentFontStyle componentFontName	black 14 plain	用于组件名称
componentStereotypeFontColor componentStereotypeFontSize componentStereotypeFontStyle componentStereotypeFontName	black 14 italic	用于组件中的构造类型
componentArrowFontColor componentArrowFontSize componentArrowFontStyle componentArrowFontName	black 13 plain	用于组件图中的箭头之上的文字

noteFontColor noteFontSize noteFontStyle noteFontName	black 13 plain	用于除序列图之外的其他图示中注释
packageFontColor packageFontSize packageFontStyle packageFontName	black 14 plain	用于报名和分区名
sequenceActorFontColor sequenceActorFontSize sequenceActorFontStyle sequenceActorFontName	black 13 plain	用于序列图中的角色
sequenceDividerFontColor sequenceDividerFontSize sequenceDividerFontStyle sequenceDividerFontName	black 13 bold	用于序列图中 divider 上的文字
sequenceArrowFontColor sequenceArrowFontSize sequenceArrowFontStyle sequenceArrowFontName	black 13 plain	用于序列图中箭头之上的文字
sequenceGroupingFontColor sequenceGroupingFontSize sequenceGroupingFontStyle sequenceGroupingFontName	black 11 plain	用于序列图中” else “上的文字
sequenceGroupingHeaderFontColor sequenceGroupingHeaderFontSize sequenceGroupingHeaderFontStyle sequenceGroupingHeaderFontName	black 13 plain	用于序列图中” alt/opt/loop “标头的文字
sequenceParticipantFontColor sequenceParticipantFontSize sequenceParticipantFontStyle sequenceParticipantFontName	black 13 plain	用于序列图中参与者的文字
sequenceTitleFontColor sequenceTitleFontSize sequenceTitleFontStyle sequenceTitleFontName	black 13 plain	用于序列图的标题
titleFontColor titleFontSize titleFontStyle titleFontName	black 18 plain	用于除序列图之外的图示的标题
stateFontColor stateFontSize stateFontStyle stateFontName	black 14 plain	用于状态图的状态
stateArrowFontColor stateArrowFontSize stateArrowFontStyle stateArrowFontName	black 13 plain	用于状态图中箭头之上的文字
stateAttributeFontColor stateAttributeFontSize stateAttributeFontStyle stateAttributeFontName	black 12 plain	用于状态图中的状态描述
usecaseFontColor usecaseFontSize usecaseFontStyle usecaseFontName	black 14 plain	用于用户案例图中的 usecase 标签

usecaseStereotypeFontColor usecaseStereotypeFontSize usecaseStereotypeFontStyle usecaseStereotypeFontName	black 14 italic	用于用户案例中的构造类型
usecaseActorFontColor usecaseActorFontSize usecaseActorFontStyle usecaseActorFontName	black 14 plain	用于用户案例图中的角色标签
usecaseActorStereotypeFontColor usecaseActorStereotypeFontSize usecaseActorStereotypeFontStyle usecaseActorStereotypeFontName	black 14 italic	用于角色的构造类型
usecaseArrowFontColor usecaseArrowFontSize usecaseArrowFontStyle usecaseArrowFontName	black 13 plain	用于用户案例图中箭头之上的文字
footerFontColor footerFontSize footerFontStyle footerFontName	black 10 plain	用于脚注
headerFontColor headerFontSize headerFontStyle headerFontName	black 10 plain	用于标头

12.5 黑白色

可以使用 `skinparam monochrome true` 命令强制使用黑白色进行输出。

```
@startuml
skinparam monochrome true

actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C

User -> A: DoWork
activate A

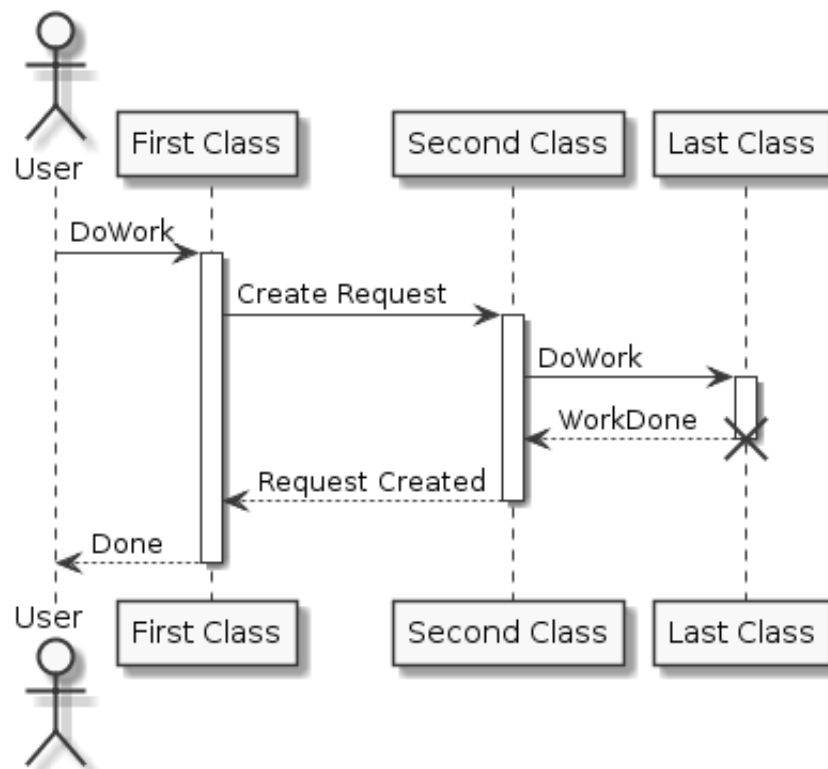
A -> B: Create Request
activate B

B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



13 预处理

PlantUML 包含少量的预处理功能，该功能对所有类型的图示都可用。

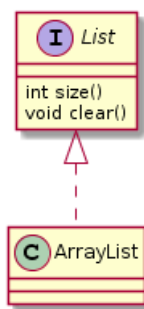
预处理功能跟 C 语言的预处理非常类似，只是把 # 换成 ! 即可。

13.1 引入文件

用 `!include` 指令引入文件。

比如你可能有一个类出现在多个图示中，这时你可以定义一个文件保存该类的描述，然后在其他图示描述文件中引入该文件而不用重复描述这个类。

```
@startuml
!include List.iuml
List <|.. ArrayList
@enduml
```



File List.iuml: interface List List : int size() List : void clear()

文件 `List.iuml` 可以包含进多个图示描述中，一旦这个文件被修改，所有包含它的图示描述中都会作相应的改动。

你可以在被引入的文件中声明多个 `@startuml/@enduml` 文本块，引入该文件时使用 `!0` 语法来指定引入哪个 `@startuml/@enduml` 文本块，`0` 是文本块的编号。

举个例子，`!include foo.txt!1` 表示 `foo.txt` 文件中的第二个 `@startuml/@enduml` 文本块被引入。

13.2 通过 URL 引入文件

使用 `!includeurl` 指令可以从 Internet 或 Intranet 引入文件到你的图示中。

你也可以使用 `!includeurl http://someurl.com/mypath!0` 来指定 `http://someurl.com/mypath` 中引入某个 `@startuml/@enduml` 文本块，`!0` 表示第一个。

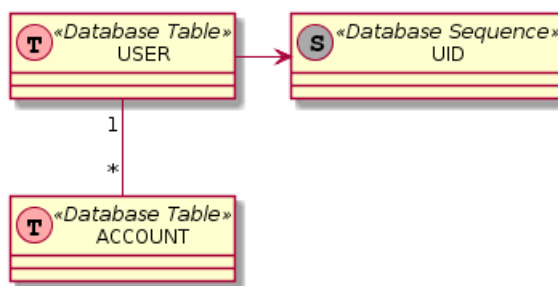
13.3 定义常量

可以使用 `!define` 指令定义常量，就像在 C 语言中定义常量一样，它只能包含字母、下划线或数字，并且不能以数字作为开头。

```
@startuml

!define SEQUENCE (S,#AAAAAA) Database Sequence
!define TABLE (T,#FFAAAA) Database Table

class USER << TABLE >>
class ACCOUNT << TABLE >>
class UID << SEQUENCE >>
USER "1" -- "*" ACCOUNT
USER -> UID
@enduml
```



当然你也可以把所有的常量在一个文件中定义好，然后用 `!include` 指令引入到图示的描述中。

常量可以用 `!undef XXX` 指令取消。

你也可以在命令行中用 `-D` 指定常量。

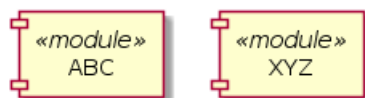
```
java -jar plantuml.jar -DTITLE="My title" atest1.txt
```

注意 `-D` 一定要放在 `-jar plantuml.jar` 之后。

13.4 宏定义

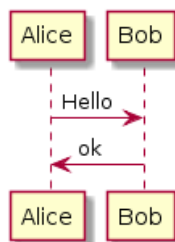
你可以定义带参数的宏。

```
@startuml
!define module(x) component x <<module>>
module(ABC)
module(XYZ)
@enduml
```



宏可以带有多个参数。

```
@startuml
!define send(a,b,c) a->b : c
send(Alice, Bob, Hello)
send(Bob, Alice, ok)
@enduml
```

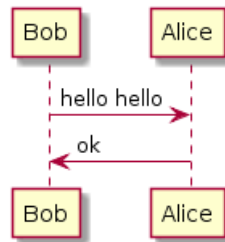


13.5 多行宏

可以使用 `!definelong` 和 `!enddefinelong` 定义跨多行的宏。

```
@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
@enduml
```

```
!enddefinelong
AUTHEN(Bob,Alice)
@enduml
```



13.6 条件判断

使用 `!ifdef XXX` 和 `!endif` 指令进行有条件的绘制。

只有当 `!ifdef` 指令后的常量被定义之后，两个指令之间的命令才会有效。

还可以添加 `!else` 部分，当常量没有被定义的时候这部分生效。

```
@startuml
!include ArrayList.iuml
@enduml
```

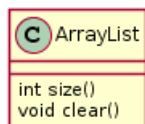


File `ArrayList.iuml`:

```
class ArrayList
!ifdef SHOW_METHODS
ArrayList : int size()
ArrayList : void clear()
!endif
```

之后就可以用 `!define` 指令激活条件判断部分。

```
@startuml
!define SHOW_METHODS
!include ArrayList.iuml
@enduml
```



还可以使用 `!ifndef` 指令，含义跟 `!ifdef XXX` 正好相反。

13.7 搜索路径

可以在命令行指定 `plantuml.include.path` 属性。

例如:

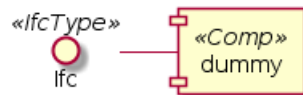
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

注意 `-D` 选项必须在 `-jar` 选项之前，如果 `-D` 位于 `-jar` 之后，那么 `-D` 选项将会被用于定义 `plantuml` 的预处理常量。

13.8 高级特效

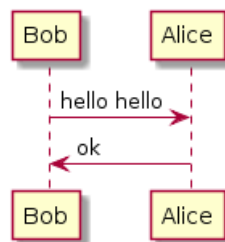
可以使用 `##` 在宏参数之后追加文字。

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



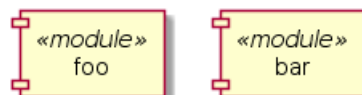
还可以在宏定义中定义其他宏。

```
@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
!enddefinelong
AUTHEN(Bob,Alice)
@enduml
```



宏支持根据参数个数的多态定义。

```
@startuml
!define module(x) component x <<module>>
!define module(x,y) component x as y <<module>>
module(foo)
module(bar, barcode)
@enduml
```



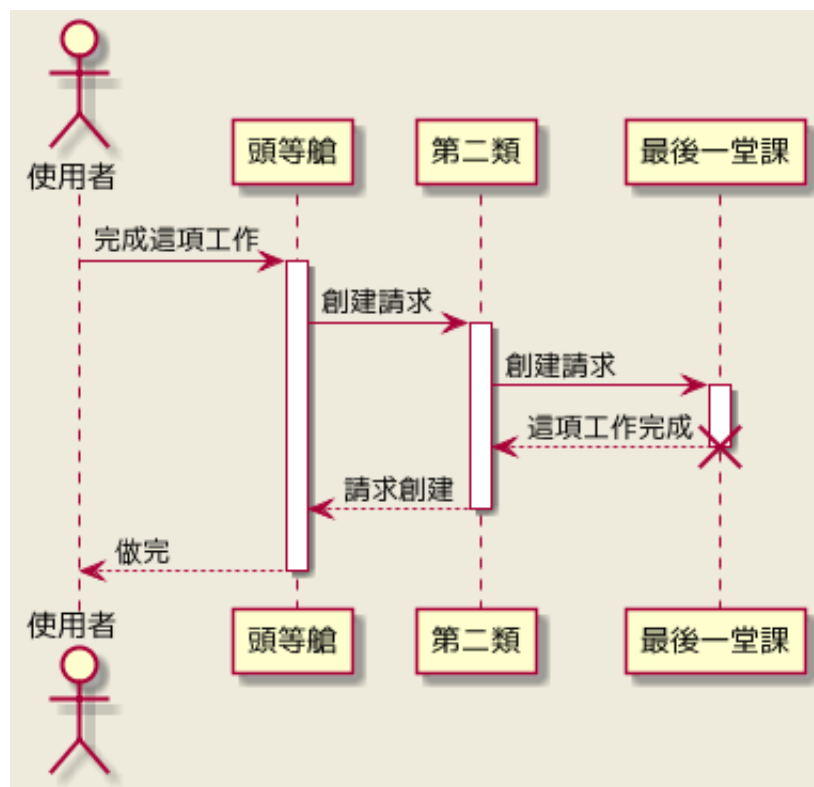
当使用文件引入的时候可以使用系统环境变量或者预定义常量。

```
!include %windir%/test1.txt
!define PLANTUML_HOME /home/foo
!include PLANTUML_HOME/test1.txt
```

14 国际化

PlantUML 语言使用字符定义角色、使用案例等待，但是字符并不仅限于 A-Z 的拉丁字符，它可以是来自任何语言的字符。

```
@startuml
skinparam backgroundColor #EEEEBD
actor 使用者
participant "頭等艙" as A
participant "第二類" as B
participant "最後一堂課" as 別的東西
使用者 -> A: 完成這項工作
activate A
A -> B: 創建請求
activate B
B -> 別的東西: 創建請求
activate 別的東西
別的東西 --> B: 這項工作完成
destroy 別的東西
B --> A: 請求創建
deactivate B
A --> 使用者: 做完
deactivate A
@enduml
```



14.1 字符集

读取 UML 描述文件时使用系统默认的字符集。一般情况下这没有问题，但是有时你可能想使用其他字符集，如使用如下命令：

```
java -jar plantuml.jar -charset UTF-8 files.txt
```

或者使用 ant 任务：

```
<target name="main">  
<plantuml dir="./src" charset="UTF-8" />  
</target>
```

下面是可用的字符集 (这依赖于你的 Java 安装情况): ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.

15 可用的色彩名称

以下是 PlantUML 中可用的色彩名称. 这些名称是大小不敏感的。

	AliceBlue		GhostWhite		NavajoWhite
	AntiqueWhite		GoldenRod		Navy
	Aquamarine		Gold		OldLace
	Aqua		Gray		OliveDrab
	Azure		GreenYellow		Olive
	Beige		Green		OrangeRed
	Bisque		HoneyDew		Orange
	Black		HotPink		Orchid
	BlanchedAlmond		IndianRed		PaleGoldenRod
	BlueViolet		Indigo		PaleGreen
	Blue		Ivory		PaleTurquoise
	Brown		Khaki		PaleVioletRed
	BurlyWood		LavenderBlush		PapayaWhip
	CadetBlue		Lavender		PeachPuff
	Chartreuse		LawnGreen		Peru
	Chocolate		LemonChiffon		Pink
	Coral		LightBlue		Plum
	CornflowerBlue		LightCoral		PowderBlue
	Cornsilk		LightCyan		Purple
	Crimson		LightGoldenRodYellow		Red
	Cyan		LightGreen		RosyBrown
	DarkBlue		LightGrey		RoyalBlue
	DarkCyan		LightPink		SaddleBrown
	DarkGoldenRod		LightSalmon		Salmon
	DarkGray		LightSeaGreen		SandyBrown
	DarkGreen		LightSkyBlue		SeaGreen
	DarkKhaki		LightSlateGray		SeaShell
	DarkMagenta		LightSteelBlue		Sienna
	DarkOliveGreen		LightYellow		Silver
	DarkOrchid		LimeGreen		SkyBlue
	DarkRed		Lime		SlateBlue
	DarkSalmon		Linen		SlateGray
	DarkSeaGreen		Magenta		Snow
	DarkSlateBlue		Maroon		SpringGreen
	DarkSlateGray		MediumAquaMarine		SteelBlue
	DarkTurquoise		MediumBlue		Tan
	DarkViolet		MediumOrchid		Teal
	Darkorange		MediumPurple		Thistle
	DeepPink		MediumSeaGreen		Tomato
	DeepSkyBlue		MediumSlateBlue		Turquoise
	DimGray		MediumSpringGreen		Violet
	DodgerBlue		MediumTurquoise		Wheat
	FireBrick		MediumVioletRed		WhiteSmoke
	FloralWhite		MidnightBlue		White
	ForestGreen		MintCream		YellowGreen
	Fuchsia		MistyRose		Yellow
	Gainsboro		Moccasin		

Contents

1	时序图	1
1.1	简单示例	1
1.2	注释	1
1.3	声明参与者	1
1.4	在参与者中使用非字符	2
1.5	消息发给自己	3
1.6	更改箭头的样式	3
1.7	修改箭头颜色	4
1.8	消息编号	4
1.9	标题 (Title)	6
1.10	给图表 (diagram) 添加备注	6
1.11	分割图表 (diagram)	7
1.12	组合消息	7
1.13	给消息添加注释	8
1.14	其他的注释	9
1.15	改变备注框的形状	10
1.16	Creole 和 HTML	10
1.17	分隔符	11
1.18	引用	11
1.19	延迟	12
1.20	空间	12
1.21	生命线的激活与撤销	13
1.22	创建参与者	14
1.23	进入和发出消息	15
1.24	构造类型和圈点	16
1.25	更多标题信息	17
1.26	包裹参与者	18
1.27	移除脚注	19
1.28	外观参数 (skinparam)	19
2	用例图	21
2.1	用例	21
2.2	角色	21
2.3	用例描述	21
2.4	基础示例	22
2.5	继承	22
2.6	使用注释	23
2.7	构造类型	24
2.8	改变箭头方向	24
2.9	添加标题	25
2.10	分割图示	25
2.11	从左向右方向	26
2.12	显示参数	27
2.13	一个完整的例子	28

3	类图	29
3.1	类之间的关系	29
3.2	关系上的标识	29
3.3	添加方法	31
3.4	定义可访问性	32
3.5	抽象与静态	33
3.6	高级类体	34
3.7	备注和模板	35
3.8	更多注释	36
3.9	Note on links	37
3.10	抽象类和接口	38
3.11	Using non-letters	39
3.12	Hide attributes, methods...	40
3.13	Hide classes	40
3.14	Use generics	41
3.15	Specific Spot	41
3.16	Packages	42
3.17	Packages style	42
3.18	Namespaces	43
3.19	Automatic namespace creation	44
3.20	Lollipop interface	45
3.21	Changing arrows direction	45
3.22	Title the diagram	46
3.23	Legend the diagram	47
3.24	Association classes	47
3.25	Skinparam	48
3.26	Skinned Stereotypes	48
3.27	Color gradient	49
3.28	Splitting large files	50
4	活动图	51
4.1	简单活动	51
4.2	箭头上的标签	51
4.3	改变箭头方向	51
4.4	分支	52
4.5	更多分支	53
4.6	同步	54
4.7	长的活动描述	54
4.8	注释	55
4.9	分区	55
4.10	给图示添加标题	56
4.11	显示参数	57
4.12	八边形活动	58
4.13	一个完整的例子	58

5	活动图 (新语法)	61
5.1	简单活动图	61
5.2	开始/结束	61
5.3	条件语句	62
5.4	重复循环	63
5.5	while 循环	63
5.6	并行处理	64
5.7	注释	65
5.8	标题和图例	65
5.9	颜色	66
5.10	箭头	66
5.11	组合 (grouping)	67
5.12	泳道 (Swimlanes)	68
5.13	分离 (detach)	69
5.14	特殊领域语言 (SDL)	70
5.15	一个完整的例子	71
6	组件图	73
6.1	组件	73
6.2	接口	73
6.3	基础的示例	73
6.4	使用注释	74
6.5	组合组件	74
6.6	改变箭头方向	75
6.7	添加图示标题	76
6.8	使用 UML2 标记符	77
6.9	Individual colors	77
6.10	显示参数	77
7	状态图	79
7.1	简单状态	79
7.2	合成状态	79
7.3	长名字	80
7.4	并发状态	81
7.5	箭头方向	82
7.6	注释	83
7.7	更多注释	84
7.8	显示参数	84
8	对象图	86
8.1	对象的定义	86
8.2	对象之间的关系	86
8.3	添加属性	86
8.4	类图中的通用特性	87

9 通用命令	88
9.1 脚注和标头	88
9.2 缩放	88
10 Salt	89
10.1 Basic widgets	89
10.2 Using grid	89
10.3 Using separator	90
10.4 Tree widget	90
10.5 Enclosing brackets	90
10.6 Adding tabs	90
10.7 Using menu	91
10.8 Advanced table	92
11 Creole	93
11.1 Emphasized text	93
11.2 List	93
11.3 Escape character	93
11.4 Horizontal lines	94
11.5 Headings	94
11.6 Legacy HTML	94
11.7 Table	95
11.8 Tree	95
11.9 Special characters	95
11.10OpenIconic	95
11.11Defining and using sprites	97
11.12Encoding Sprite	97
11.13Importing Sprite	97
11.14Examples	97
12 修改字体和颜色	99
12.1 使用方法	99
12.2 嵌套	99
12.3 颜色	100
12.4 字体的颜色、名字和尺寸	101
12.5 黑白色	104
13 预处理	105
13.1 引入文件	105
13.2 通过 URL 引入文件	105
13.3 定义常量	105
13.4 宏定义	106
13.5 多行宏	106
13.6 条件判断	107
13.7 搜索路径	107
13.8 高级特效	108

14 国际化	109
14.1 字符集	109
15 可用的色彩名称	111