Ansible & Jinja2

Le moteur de template

Arnaud Blancher - AdminSys @ Orange - baladestrash @ gmail[.]com
Meetup Ansible Paris - 29/03/2016

Notation

```
liste = [ 'élément 1', "élément 2", 3 ]
dictionnaire = {
 "cle1": 'valeur 1',
 'cle2' : 'valeur 2'
}
tuple = ( 'élément 1', "élément 2", 3)
# Un tuple est une liste non modifiable
```

{{ ... }} et {% ... %}

{{ ... }} : afficher la valeur d'une expression (variable)
{% ... %} : exécuter une instruction

{{ ... }} : lire

```
{{ ma_var }}
{{ foo.bar }}
{{ foo['bar'] }}
```

{% ... %} exécuter

{% ... %} set variable

```
{% set navigation = [ ('index.html', 'Index'), ('about.html', 'About') ] %}
{% for l in navigation %}
  *{{ l[1] }} : {{ l[0]}}*
{%-  endfor %}
```

{% ... %} macro

```
{% macro macro_doubler(nombre=0) -%}
{{2*nombre}}
{%- endmacro %}
```

```
{%- set ledouble = macro_doubler(5) %}
{{ledouble}}
```

commentaire

```
{#
un commentaire
#}
```

commentaire

raw : escape

via la commande {% raw %} ... {% endraw %}

```
{% raw %}
template et {{variable}}
 à ne pas interpréter
{% endraw %}
```

imprimer un bout de commande

```
{{ '{{' }}
```

Espaces dans les templates

Le saut de ligne qui suit immediatement le %} est supprimé du rendu.
Les espaces, tabulations et autres saut de lignes sont conservés tel quel.

{%- à la place de {% permet de supprimer les espaces et tabulations à gauche de l'accolade.

```
---
{% for ligne in liste1 -%}

{{ ligne.title }}

{%- endfor %}
--
```

```
--
foo
bar
py
--
```

block et extends

Un fichier template peut contenir des blocs qui seront ou non remplacés par d'autres blocs d'un autre template (en plus des variables)
Le template original :

```
Ligne 1
{% block section31 %}
Le contenu par défaut de la section31  {{ var1 }}
{% endblock %}

Ligne 2
{% block C %}
Le contenu par défaut du bloc C {{ var2 }}
{% endblock C %}
```

block et extends

Surcharge du template original :

```
{% extends "modele" %}

{% block section31 %}
Nouvelle version de la section31 {{ var3 }}
{% endblock %}
```

{% extends "modele" %} utilise le template original "modele" (fichier)

Seul les {% block .. %}...{% endblock %} présent dans ce fichier redéfinissent ceux du template original.

La sortie :

```
Ligne 1
Nouvelle version de la section31 trois

Ligne 2
Le contenu par défaut du bloc C deux
```

block et extends

```
{% extends "modele" %}

{% block section42 %}
{{ super() }}
Ajout dans le bloc !
{% endblock %}
```

super() duplique le contenu du bloc initial et permet d'ajouter des lignes avant et/ou après le bloc.

Surcharge dans un role avec extends

```
roles:
- { role: extends_block-pere, conf: etc/monapp/conf2 }
```

```
# roles/extends_block-pere/tasks/main.yml
- name: "modele avec block et extends"
  template: src={{ conf }} dest=/tmp/conf
```

```
# roles/extends_block-pere/templates/etc/monapp/conf2
{% extends "modele"  %}
...
```

Il FAUT que 2 les fichiers (l'appelant et le référencé par extends) soient dans le même répertoire,
mais les liens symboliques fonctionnent.
La variable qui définit le nom du fichier est relative au répertoire templates (qui semble hard codé dans Ansible 1.9.4).

if else

```
{% if var1 %}
  ...
{% elif var2 != "valeur" %}
  ...
{% else %}
  ...
{% endif %}
```

for .iteritems() : itérer sur un dictionnaire

```
# for.yml
  ...
  vars:
    dict1: {
      'cle1': 'val1',
      'cle2': 'val2'
    }
  ..
```

```
# templates/for_dict.j2
{% for cle,valeur in dict1.iteritems() %}
  la clé {{ cle }} à pour valeur {{ valeur }}
{% endfor %}
```

for .iteritems() if : partie d'arbre et condition

```yaml
# ./for.yml
...
vars:
  dict100:
    dict110:
      clea: val a
      cleb: val b   # oui
    dict120:        # non car vide
    dict130:
      cleb: val 130 # oui
    dict140:
      clea: val *a
      cleb:         # non car vide
      cled: val *d
...
```

```
# templates/for_dict.j2
{% for cle,valeur in dict100.iteritems() %}
  {# pour les "cle"s qui ont une valeur non vide #}
  {% if valeur %}
    la clé *{{ cle }}* à pour valeur {{ valeur }}
    {% for cle2,valeur2 in valeur.iteritems() %}

      {# filtrage sur cleb qui ont des valeurs non vide #}
      {% if cle2=="cleb" and valeur2 %}
      la clé {{ cle2 }} à pour valeur {{ valeur2 }}
      {% endif %}

    {% endfor %}
  {% endif %}
{% endfor %}
```

for in : itérer sur une liste de dictionnaire

```
vars:
  liste1: [ { title: "foo", id: 1 },
              { title: "bar", id: 2} ]
```

```
# templates/for.j2
{% for dict in liste1 %}
  * {{ dict.title }} {{ dict.id }}
{% else %}
  * Cela s'affichera si liste1 est définie mais sans élément (ie liste1: [] ) *
{% endfor %}
```

for ... in ... recursive : boucler recursivement

```
# recursive.yml
...
sitemap: [
  { href: h100, title: t100,
    enfant: [ { href: h110, title: t110 } ]
  },
  { href: h200, title: t200 },
  { href: h300, title: t300 },
  { href: h400, title: t400,
    enfant: [
      { href: h410, title: t410 },
      { href: h420, title: t420 }
    ]
  },
  { href: h500, title: t500 }
]
```

'recursive' permet de rappeler la boucle externe via 'loop'

```
# templates/recursive.j2
{% for item in sitemap recursive %}
  * {{ item.title }} : {{ item.href } }
  {% if item.enfant is defined %} (
    {{ loop(item.enfant) }} )
  {% endif %}
{% endfor %}
```

macro

```
# macro.yml
 ...
  vars:
     f1_type: "text"
     f1_name: "commentaire"
     f1_value: "votre com"
     f1_size: "50"
  ...
```

La macro s'appelle alors par son nom comme si c'était une fonction.

```
 # templates/macro.j2
{% macro input(name, value='', type='text', size=20) -%}
    <input type="{{ type }}" name="{{ name }}" value="{{
        value|e }}" size="{{ size }}">
{%- endmacro %}

{ input(f1_name, type=f1_type, value=f1_value, size=f1_size) }}
```

import de macro

import fonctionne comme en python

import complet

```
{% import 'macros.txt' as macros with context %}
# Les macros de 'macros.txt' seront alors accessible via
{{ macros.macro1(...) }}
```

import ciblé

```
{% from 'macros.txt' import macro1 as macro1_1, macro2  with context%}
# Les macros de 'macros.txt' seront alors accessible via
{{ macro1_1(...) }}
{{ macro2(...) }}
```

Ne pas oublier le 'with context', sinon erreur
"AttributeError: 'NoneType' object has no attribute 'add_locals'" (ansible v2.0.0.2)

| : filtre

Les filtres appliquent un retraitement sur les variables. Ils se chaînent via le signe |

```
{{ liste2 | join('/') }} # joindre les éléments en utilisant le séparateur '/'
{{ texte | replace("un ", "voici un ") | capitalize }}
{{ ansible_managed | replace("/home/{{uid}}", '.') }}
"{{ groups.dba | union(groups.dbb) }}" # union des groupes d'inventaire 'dba' et 'dbb'
"{{ database_host | default('localhost') }}" # ou bien | d('localhost')
```

Les filtres sont aussi utilisable dans les templates sur la totalité d'un bloc :

```
{% filter upper %}
  Passer ce texte
  en majuscule.
{% endfilter %}
```

include (template)

```
{% include 'include.inc1.j2' %}
```

Include en ignorant l'erreur si le fichier n'existe pas

```
{% include 'include.inc1.j2' ignore missing %}
```

Inclure le 1er fichier trouvé depuis une liste

```
{% include ['include.inc3.j2','include.inc2.j2','include.inc1.j2']  %}
```

Inclure un fichier préferentiellement en fonction d'une variable

```
{% set fichier =  ['include.inc',dc,'.j2']| join("")  %}
{% include [fichier,'include.inc1.j2']  %}
```

Merci