w.c.s. - Development #21545

permettre des conditions utilisant le format Django

31 janvier 2018 09:19 - Thomas Noël

Statut:	Fermé	**Début:**	31 janvier 2018
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	

Description

On pourrait avoir des conditions (de page, de saut) qui soient des expressions Django, celles qu'on utilise dans les {% if ... %}

Pour les évaluer, on ferait quelque chose comme '{% load qommon %}{% if %s %}OK{% else %}KO{% endif %}' % condition et en fonction de OK ou KO ou erreur, on réagirait.

Avantage :

- la même doc pour les conditions de templates et conditions
- la simplicité des filtres "à la SPIP" qui se lit plus facilement, genre form_var_data|age_in_days > 10 plus clair que utils.age_in_days(form_var_data) > 10 à mon avis (mais surtoutn c'est pareil que dans les templates).

Inconvénient :

- historique à gérer. Pour ça, j'imagine un widget "ConditionWidget" qui aurait un input text (l'actuelle condition) et un choix condition_type "Django" ou "Python". Ce choix serait à "Django" par défaut, sauf pour les conditions existantes où ça serait None qui deviendrait Python. Reste à voir l'UI...

Voilà, ce ticket pour avis et commentaires...

Demandes liées:

Lié à w.c.s. - Development #21550: Stocker les conditions sous forme de dict...	**Fermé**	**31 janvier 2018**
Lié à w.c.s. - Development #19752: Possibilité d'utiliser des champs (et non ...	**Fermé**	**29 octobre 2017**
Lié à w.c.s. - Development #22878: Obtenir les exceptions lors de l'évaluatio...	**Fermé**	**29 mars 2018**

Révisions associées

Révision a71a61a1 - 05 avril 2018 16:33 - Frédéric Péters

general: factor out page and jump condition evaluation (#21545)

Révision 98b2ba26 - 05 avril 2018 16:33 - Frédéric Péters

general: add evaluation of django conditions (#21545)

Révision 24e64134 - 05 avril 2018 16:33 - Frédéric Péters

general: add condition type selection to widget (#21545)

Historique

#1 - 31 janvier 2018 09:25 - Frédéric Péters

- Description mis à jour

#2 - 31 janvier 2018 09:33 - Frédéric Péters

Ça introduit la zone grise "quand ni OK ni KO" n'est obtenu; je voyais d'abord ça comme un problème (et j'allais suggérer de juste considérer OK et tout le reste pas bon) mais dans un second temps je me dis que ça peut être pas mal d'exiger ces valeurs, ça permet de détecter/signaler les moments où autre chose est obtenu.

J'avais aussi une petite crainte sur OK/KO et comment ça va s'ajouter sur des situations où on hésite déjà à savoir si la condition doit être rencontrée ou pas (dans les conditions de sortie de page). Mais ça pourrait au contraire même clarifier les choses (si le résultat est KO alors le message d'erreur est affiché).

Cela étant j'hésiterais encore sur le sujet, le fonctionnement quelque chose = vrai, vide = faux, est plus proche de ce qu'on a l'habitude de faire et se

prête aussi bien aux templates (tester qu'un champ est rempli {{form_var_plop}} plutôt que {% if form_var_plop }OK{ else }KO{ endif %}. Mais à nouveau ici être explicite est sans doute mieux qu'être concis.

#3 - 31 janvier 2018 10:05 - Frédéric Péters

- Description mis à jour

#4 - 31 janvier 2018 10:08 - Frédéric Péters

En fait, j'étais perdu à cause de {% load qommon %}{% if %s %}OK{% else %}KO{% endif %} j'avais pris le %s comme un % de trop, perdu là par hasard (et "corrigé" puis annulé la modif dans la description); mon commentaire était donc à côté du sujet…

#5 - 31 janvier 2018 10:41 - Benjamin Dauvergne

Moi ce qui me plaît surtout ici c'est d'avoir un/des champ/s condition/templates qui affichent clairement le format, Template EZT, Template Django, Python. Je pense qu'en lui même ce point peut faire l'objet d'un ticket, une fois qu'on a ça, avoir 1 ou 1000 formats, des formats progressivement dépréciés, aucun souci. Ça évitera les bidouilles de "ça parse en Django, ok c'est du Django, ça parse en EZT, ok c'est de l'EZT, etc..". La migration consistera à appliquer la bidouille pour retrouver le vrai format et conserver cette information.

#6 - 31 janvier 2018 10:59 - Frédéric Péters

> un/des champ/s condition/templates qui affichent clairement le format

C'est un autre ticket à créer, sur le même mode que #19112 qui demande la même chose pour les champs pouvant être calculés.

#7 - 31 janvier 2018 12:35 - Frédéric Péters

- Lié à Development #21550: Stocker les conditions sous forme de dictionnaire ajouté

#8 - 22 mars 2018 12:47 - Frédéric Péters

- Lié à Development #19752: Possibilité d'utiliser des champs (et non plus des pages) conditionnels ajouté

#9 - 24 mars 2018 10:25 - Frédéric Péters

En aparté.

Pour me faire une meilleure idée des conditions actuelles j'ai extrait du SaaS de prod toutes les conditions (pages et sauts), ça en fait 3001, 1525 différentes, la plus courte compte 5 caractères (False), la plus longue 4190 (…); la moyenne est à 69, la médiane à 47. Il y a 1104 conditions uniques utilisant l'opérateur ==, 89 utilisant != et 9 utilisant <>, 150 in, 32 not in. 42 appels à vars(), 0 à locals(), 0 à globals(). 9 upper() et 2 lower(). 21 utilisent datetime (de différentes manières). 245 and et 246 or. 71 les deux en même temps.

L'utilisation la plus commune qui ne marchera pas avec Django, c'est les conditions type form_var_xxx in ('option 1', 'option 2') (qui se trouve cependant déjà souvent exprimée sous forme de form_var_xxx 'option 1' or form_var_xxx 'option 2') (et c'est bien sûr ainsi qu'on arrive à 4190 caractères).

#10 - 24 mars 2018 13:34 - Frédéric Péters

- Fichier 0001-general-factor-out-page-and-jump-condition-evaluatio.patch ajouté

- Fichier 0002-general-add-evaluation-of-django-conditions-21545.patch ajouté

- Fichier 0003-general-add-condition-type-selection-to-widget-21545.patch ajouté

- Statut changé de Nouveau à En cours

- Patch proposed changé de Non à Oui

- 0001 pour partager le code d'évaluation des conditions entre les conditions (entrée/sortie) de page et les sauts.
- 0002 pour ajouter la prise en charge de conditions Django.
- 0003 pour modifier le widget des conditions, pour permettre la sélection entre les deux types de conditions.
- et à côté, dans #22102 0004, pour la validation à la volée des conditions.

C'est aussi dans https://git.entrouvert.org/wcs.git/log/?h=wip/21545-django-conditions

#11 - 24 mars 2018 14:33 - Benjamin Dauvergne

En aparté aussi (et en vrac).

Je ne suis toujours pas persuadé que ce soit un problème de format ou de langage (bien sûr on peut simplifier, faire que *getitem* ou *getattr* soient identiques au niveau syntaxe), j'étais resté pour ma part sur l'idée qu'on manquait simplement d'aide à validation des conditions/expressions à base de warning et pas d'erreurs de validation, le fait de bloquer complètement la création d'une condition/expression nous limite à de la validation basique

(ça parse), si on se permet des warning on peut aller plus loin. Aussi on ne travaille pas assez les possibilités offertes par l'interpréteur/parser inclu dans Python lui même qui permet beaucoup de choses.

1. on a beaucoup trop de variables globales et de communications via ces variables globales, raisonner sur l'état du système au moment d'une évaluation n'est pas évident même pour nous, exemple: les appels de WS nommés et les data_source font souvent référence de manière implicite à des variables qui devront exister au moment où on les appelle, ce serait beaucoup plus simple de raisonner dessus si ça devenait des fonctions et pas des variables (à Nanterre au lieu de webservice.rsu_reseau on aurait webservice.rsu_reseau(rsu_id=session_var_rsu_id); en plus ça rend tous ces objets peu réutilisables (si on doit appeler 3 fois le même web-service avec des paramètres différents on doit soit le définir 3 fois, soit avoir une expression compliquée vars().get('var1') or vars().get('var2') en espérant que jamais les 3 ne soient définies en même temps).

2. on a pas de validation avancée parce qu'on a pas la liste des variables qui nous intéressent. On pourrait l'avoir au moins en partie et accepter un peu d'incertitude sur le reste. C'est un travail en suspend via register_dynamic_sources() mais il manque des sources possibles, wscalls, data_source, scripts, session_var_* et pour la source FormData on a pas vraiment la listes des variables du formulaire sur lequel on travaille, cette partie n'est qu'ébauchée[1] il faudrait voir comment obtenir dynamiquement le/les FormData courant et le Workflow courant (en feedant des classes plutôt que des instances et en ayant un get_context_schema() qui retourne une description des variables accessibles), on peut même l'utiliser pour ajouter du typeahead dans les champs d'expression.

3. une fois qu'on a un dictionnaire complet des variables accessible on peut commencer à valider simplement qu'on utilise pas de variables non définis (à minima la variable peut exister, peut-être que dans un contexte précis elle n'existera pas, ex.: session_var_, mais on a un début de validation), ensuite on peut aller vers de la validation statique plus compliquée à la mypy/py3 (vars().get('session_var_x') c'est ok ça ne foirera jamais, session_var_x ça ne l'est pas) en laissant les sources nous retourner en plus du nom et de la catégorie un type abstrait (session_var_x : Or(Undefined, Str), form_var_x_not_required : Or(None, Str), vars : None -> Dict, .get : Dict, Str -> Any) on pourra ainsi lever des warnings de plus en plus fin en affinant notre interpréteur abstrait.

4. comme dans Jenkins ou Drupal on devrait avoir des notifications des warnings actuels, sans avoir à visiter la totalité des formulaires et workflow ou attendre une exception.

5. si on part sur du Django on repart de zéro parce que Django ne fournit pas les outils équivalents au module ast inclu dans Python (on peut assez facilement et automatiquement transformer x.y en resolve(x, 'y') ou resolve se comporterait comme dans les templates Django, idem pour remplacer toutes les variables indéfinis par None ou autre, c'est vraiment plus simple de travailler sur un AST python que sur le résultat du parsing des template Django)

[1]

```
    @classmethod
    def get_substitution_variables_list(cls):
        variables = []
        # we can't advertise fields, as this is a metaclass that will be used
        # in FormDef.data_class() to create a real class
        for field in []: # cls.formdef.fields:
            # we only advertise fields with a varname, as they can be
            # considered stable
            if field.varname:
                variables.append((
                        _('Form'), 'form_var_'+field.varname,
                        _('Form Field: %s') % field.label))
        user_variables = get_publisher().user_class.get_substitution_variables_list(prefix='form_')
        for cat, name, comment in user_variables:
            variables.append((_('Form'), name, _('Form Submitter Field')))
        return variables
```

#12 - 24 mars 2018 14:45 - Pierre Cros

Je risque pas de participer au débat technique bien entendu, mais retour d'expérience quand même, vous en ferez ce que vous voudrez.

Benjamin Dauvergne a écrit :

> Je ne suis toujours pas persuadé que ce soit un problème de format ou de langage (bien sûr on peut simplifier, faire que *getitem* ou *getattr* soient identiques au niveau syntaxe), j'étais resté pour ma part sur l'idée qu'on manquait simplement d'aide à validation des conditions/expressions à base de warning et pas d'erreurs de validation, le fait de bloquer complètement la création d'une condition/expression nous limite à de la validation basique (ça parse), si on se permet des warning on peut aller plus loin. Aussi on ne travaille pas assez les possibilités offertes par l'interpréteur/parser inclu dans Python lui même qui permet beaucoup de choses.

Pourquoi pas mais je crois qu'on a eu du warning à un moment (machin orange), perso ça ne m'a pas aidé beaucoup : j'ai pas le niveau. Et comme le but premier c'est je crois de ne pas recevoir d'erreur, bloquer c'est pas mal. "Aller plus loin" pour que ce soit utile pour moi, ce serait vraiment aller beaucoup plus loin je crois.

> 2. on a pas de validation avancée parce qu'on a pas la liste des variables qui nous intéressent.

Oui là je suis d'accord, avoir la liste des variables le plus souvent possible, et idéalement pouvoir les sélectionner pour éviter les fautes de frappes, ce serait un gros plus (même si je comprends rien à ton plan en la matière, j'imagine bien que c'est complexe).

#13 - 24 mars 2018 15:16 - Benjamin Dauvergne

Frédéric Péters a écrit :

> En aparté.
>
> Pour me faire une meilleure idée des conditions actuelles j'ai extrait du SaaS de prod toutes les conditions (pages et sauts), ça en fait 3001, 1525 différentes, la plus courte compte 5 caractères (False), la plus longue 4190 (…); la moyenne est à 69, la médiane à 47. Il y a 1104 conditions uniques utilisant l'opérateur ==, 89 utilisant != et 9 utilisant <>, 150 in, 32 not in. 42 appels à vars(), 0 à locals(), 0 à globals(). 9 upper() et 2 lower(). 21 utilisent datetime (de différentes manières). 245 and et 246 or. 71 les deux en même temps.
>
> L'utilisation la plus commune qui ne marchera pas avec Django, c'est les conditions type form_var_xxx in ('option 1', 'option 2') (qui se trouve cependant déjà souvent exprimée sous forme de form_var_xxx 'option 1' or form_var_xxx 'option 2') (et c'est bien sûr ainsi qu'on arrive à 4190 caractères).

Tu pourrais attacher la liste au ticket ?

#14 - 24 mars 2018 15:20 - Benjamin Dauvergne

Pierre Cros a écrit :

> Pourquoi pas mais je crois qu'on a eu du warning à un moment (machin orange), perso ça ne m'a pas aidé beaucoup : j'ai pas le niveau. Et comme le but premier c'est je crois de ne pas recevoir d'erreur, bloquer c'est pas mal. "Aller plus loin" pour que ce soit utile pour moi, ce serait vraiment aller beaucoup plus loin je crois.

Ce n'est pas possible il y a des choses qui sont bloquantes (ce n'est pas la bonne syntaxe) et d'autres pas (possible que session_var_x n'existe pas, je n'en suis pas sûr, c'est dynamique).

> 2. on a pas de validation avancée parce qu'on a pas la liste des variables qui nous intéressent.

> Oui là je suis d'accord, avoir la liste des variables le plus souvent possible, et idéalement pouvoir les sélectionner pour éviter les fautes de frappes, ce serait un gros plus (même si je comprends rien à ton plan en la matière, j'imagine bien que c'est complexe).

Si on a une liste de variables sûres (possible faux positifs, session_var_x, aucun faux négatif, i.e. si la variable n'est pas dans la liste elle ne peut absolument pas être référencée), on peut mettre encore plus d'erreurs du coté bloquant, en plus de fournir une aide à la saisie.

#15 - 24 mars 2018 15:28 - Benjamin Dauvergne

Avec Django on perd toutes les erreurs NameError et AttributeError:

```
In [6]: Template('{% if zozo == 3 %}OK{% else %}KO{% endif %}').render(Context())
Out[6]: u'KO'

In [8]: Template('{% if zozo.x.y.g == 3 %}OK{% else %}KO{% endif %}').render(Context({'zozo': 3}))
Out[8]: u'KO'
```

en fait on ne recevra jamais aucune exception lors de l'interprétation d'une variable dans un template Django:

```
In [13]: def f():
    ...:     raise Exception('x')
    ...:

In [14]: Template('{% if f == 3 %}OK{% else %}KO{% endif %}').render(Context({'f': f}))
Out[14]: u'KO'
```

on ne peut pas faire d'arithmétique:

```
In [8]: Template('{% if zozo + 5 == 3 %}OK{% else %}KO{% endif %}').render(Context({'zozo': 3}))
Out[8]: TemplateSyntaxError: Could not parse the remainder: '+' from '+'
```

Tout ce qu'on gagne c'est x.y au lieu de x['y'] mais j'explique comment ajouter ça à la syntaxe Python si on le souhaite, et x|upper au lieur de x.upper() (mais tolérant aux erreurs, pas de x, x est un nombre, x est None, tout passe).

```
In [42]: Template('{% if zozo|upper %}OK{% else %}KO{% endif %}').render(Context({'zozo': 3}))
Out[42]: u'OK'

In [43]: Template('{% if zozo|upper %}OK{% else %}KO{% endif %}').render(Context({'zozo': None}))
Out[43]: u'OK'

In [44]: Template('{% if zozo|upper %}OK{% else %}KO{% endif %}').render(Context())
Out[44]: u'OK'
```

#16 - 28 mars 2018 14:59 - Frédéric Péters

On note que le comportement par défaut est pour Django d'ignorer silencieusement les erreurs dans les gabarits, il y a des pistes autour de settings.TEMPLATE_STRING_IF_INVALID et/ou monkeypatcher ignore_failures.

On y va quand même, en notant ça comme beta.

#17 - 28 mars 2018 15:50 - Frédéric Péters

Les patchs à jour dans https://git.entrouvert.org/wcs.git/log/?h=wip/21545-django-conditions (avec la mention (Beta)).

#18 - 28 mars 2018 15:58 - Benjamin Dauvergne

Ça n'aura pas d'effet sur {% if x == 3 %} uniquement sur {{ x }}, ce n'est pas le même code qui s'exécute dans les deux cas.

```
django/template/smartif.py:

def infix(bp, func):
    """
    Creates an infix operator, given a binding power and a function that
    evaluates the node
    """
    class Operator(TokenBase):
        lbp = bp

        def led(self, left, parser):
            self.first = left
            self.second = parser.expression(bp)
            return self

        def eval(self, context):
            try:
                return func(context, self.first, self.second)
            except Exception:
                # Templates shouldn't throw exceptions when rendering.  We are
                # most likely to get exceptions for things like {% if foo in bar
                # %} where 'bar' does not support 'in', so default to False
                return False

    return Operator
```

Tout est caché dès que c'est dans un if, bien sûr on peut monkeypatcher...

#19 - 28 mars 2018 16:09 - Frédéric Péters

Tu crées un ticket pour ajouter un reporting des erreurs dans l'évaluation de ces conditions ?

#20 - 28 mars 2018 21:57 - Benjamin Dauvergne

Frédéric Péters a écrit :

> Tu crées un ticket pour ajouter un reporting des erreurs dans l'évaluation de ces conditions ?

Chez Django tu veux dire ?

#21 - 29 mars 2018 09:51 - Frédéric Péters

Je pensais plutôt à notre redmine, qui pourrait pointer un ticket côté django sûr, mais pour commencer je n'étais pas trop gêné par l'idée de monkeypatcher si nécessaire. (surtout que même si ça arrive à bouger, ça sera dans une version compatible uniquement python 3 dont on est encore loin côté wcs).

#22 - 29 mars 2018 11:18 - Benjamin Dauvergne

- Lié à Development #22878: Obtenir les exceptions lors de l'évaluation d'une condition au format Django ajouté

#23 - 29 mars 2018 11:24 - Benjamin Dauvergne

Voilà #22878.

#24 - 03 avril 2018 15:59 - Thomas Noël

Relecture de la branche

- general: factor out page and jump condition evaluation (#21545)

Je ne vois pas si le *repr* va être utilisé lors du « get_publisher().notify_of_exception(sys.exc_info()) » , si non ça serait pas mal de faire un « condition

= self.value » juste avant, comme dans wcs/wf/jump.py actuellement.

Il y a aussi une condition dans WorkflowGlobalActionTimeoutTrigger::must_trigger, saurait-elle être remplacée ici ?

- general: add evaluation of django conditions (#21545)

OK

- general: add condition type selection to widget (#21545)

OK

- forms: evaluate conditions during typing (#22102)

sans doute passer par engines['django'].from_string(xxx) au lieu de Template(xxx) afin de prévoir Django 1.11 (cf #20936)

#25 - 05 avril 2018 16:01 - Frédéric Péters

> Je ne vois pas si le repr va être utilisé lors du « get_publisher().notify_of_exception(sys.exc_info()) » , si non ça serait pas mal de faire un « condition = self.value » juste avant, comme dans wcs/wf/jump.py actuellement.

Yes, elle apparait bien, genre self = <Condition (python) '1/0'> (mais je peux faire un condition = self si on trouve ça plus clair).

> Il y a aussi une condition dans WorkflowGlobalActionTimeoutTrigger::must_trigger, saurait-elle être remplacée ici ?

Je viens de regarder mais en fait non là c'est une expression devant donner comme résultat une date, pas une condition.

> sans doute passer par engines['django'].from_string(xxx) au lieu de Template(xxx) afin de prévoir Django 1.11 (cf #20936)

Vérifié ça marche bien avec Template() en Django 1.11. (pas regardé pour quoi ça avait été changé dans #20936).

Bref, pas de changement; sauf si on veut condition = self.

#26 - 05 avril 2018 16:08 - Thomas Noël

Ack donc (désolé pour WorkflowGlobalActionTimeoutTrigger::must_trigger, lu trop vite)

Pour le condition=self, c'est comme tu veux. Je m'étais habitué à cette astuce mais je peux instantanément m'habituer au *repr*, qui est plus "propre".

#27 - 05 avril 2018 16:48 - Frédéric Péters

- Statut changé de En cours à Résolu (à déployer)

> Pour le condition=self, c'est comme tu veux. Je m'étais habitué à cette astuce mais je peux instantanément m'habituer au repr, qui est plus "propre".

Et on se servira du repr pour le faire remonter/mettre en évidence plus facilement, j'ai poussé ça condition = self.

```
commit 24e64134d7db56bc3696d4da20534140ccddf99a
Author: Frédéric Péters <fpeters@entrouvert.com>
Date:   Sat Mar 24 12:15:09 2018 +0100

    general: add condition type selection to widget (#21545)

commit 98b2ba266164745321ecaf9e14035dc2003e62a4
Author: Frédéric Péters <fpeters@entrouvert.com>
Date:   Sat Mar 24 11:25:20 2018 +0100

    general: add evaluation of django conditions (#21545)

commit a71a61a18046cf8a31c5de87b191488cc1cbb811
Author: Frédéric Péters <fpeters@entrouvert.com>
Date:   Sat Mar 24 10:43:13 2018 +0100

    general: factor out page and jump condition evaluation (#21545)
```

#28 - 23 décembre 2018 14:43 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

Fichiers

0002-general-add-evaluation-of-django-conditions-21545.patch	2,5 ko	24 mars 2018	Frédéric Péters
0001-general-factor-out-page-and-jump-condition-evaluatio.patch	8,55 ko	24 mars 2018	Frédéric Péters
0003-general-add-condition-type-selection-to-widget-21545.patch	4,64 ko	24 mars 2018	Frédéric Péters