Refonte graphique des emplois

Réusiner sereinement

François FREITAG

Les emplois de l'inclusion
GIP Inclusion

7 octobre 2024

Sommaire

Plongée dans django ?

Mécanisme de vérification des variables au rendu des templates django

Intéressés ?

- **Oui** : Restez
- **Non** : ☕ [30 minutes]

Gestion des variables non définies

Qu'affiche ?

```
<div>
  {{ does_not_exist }}
</div>
```

Gestion des variables non définies

Qu'affiche ?

```
<div>
  {{ does_not_exist }}
</div>
```

```
<div>

</div>
```

Gestion des variables non définies

Qu'affiche ?

```
<div>
  {{ does_not_exist }}
</div>
```

```
<div>

</div>
```

Pourquoi ?

STRING_IF_INVALID

« Generally, if a variable doesn't exist, the template system inserts the value of the engine's `string_if_invalid` configuration option, which is set to '' (the empty `string`) by default.

Filters that are applied to an invalid variable will only be applied if `string_if_invalid` is set to '' (the empty string). If `string_if_invalid` is set to any other value, variable filters will be ignored.

This behavior is slightly different for the `if`, `for` and `regroup` template tags. If an invalid variable is provided to one of these template tags, the variable will be interpreted as `None`. Filters are always applied to invalid variables within these template tags.

If `string_if_invalid` contains a '%s', the format marker will be replaced with the name of the invalid variable. »

PYTEST-DJANGO FAIL-ON-TEMPLATE-VARS

Le harnais `pytest-django` fournit une option pour faire échouer un test lorsqu'un template utilise une variable non définie :

« `fail-on-template-vars`: fail for invalid variables in templates »

Cet utilitaire est imparfait, et a été amélioré par Xavier.

- interfère avec les `OneToOneFields`
- ignore les filtres `|default:`

Rendu des templates : moteur

```python
class Engine:  # django.template.engine
    def from_string(self, template_code):
        return Template(template_code, engine=self)

class Template:
    def __init__(self, template_string, **kwargs):
        ...
        self.source = str(template_string)
        self.nodelist = self.compile_nodelist()

    def compile_nodelist(self):
        # Slighly simplified:
        lexer = Lexer(self.source)
        tokens = lexer.tokenize()
        parser = Parser(
            tokens,
            self.engine.template_libraries,
            self.engine.template_builtins,
            self.origin,
        )
        return parser.parse()
```

Rendu des templates : parser

```python
class Parser:
    def __init__(self, tokens, **kwargs):
        # Reverse the tokens so delete_first_token(), prepend_token(), and
        # next_token() can operate at the end of the list in constant time.
        self.tokens = list(reversed(tokens))
        ...

    def compile_filter(self):
        return FilterExpression(token, self)

    def parse(self, **kwargs):
        nodelist = NodeList()
        while self.tokens:
            token = self.next_token()
            # Use the raw values here for TokenType.* for a tiny performance boost.
            token_type = token.token_type.value
            if token_type == 0:  # TokenType.TEXT
                self.extend_nodelist(nodelist, TextNode(token.contents), token)
            elif token_type == 1:  # TokenType.VAR
                if not token.contents:
                    raise self.error(token, "Empty variable tag on line %d" % token.lineno)
                try:
                    filter_expression = self.compile_filter(token.contents)
                except TemplateSyntaxError as e:
                    raise self.error(token, e)
                var_node = VariableNode(filter_expression)
                self.extend_nodelist(nodelist, var_node, token)
            elif token_type == 2:  # TokenType.BLOCK
                pass  # We can ignore the rest.
```

FilterExpression

```python
class FilterExpression:
    """Parse a variable token and its optional filters [...]"""
    # {{ user }}, {{ user | default:None }}
    def resolve(self, context, ignore_failures=False):
        ...
        try:
            obj = self.var.resolve(context)
        except VariableDoesNotExist:
            if ignore_failures:
                # FirstOfNode, ForNode, IfChangedNode, IfNode, RegroupNode.
                obj = None
            else:
                string_if_invalid = context.template.engine.string_if_invalid
                if string_if_invalid:
                    if "%s" in string_if_invalid:
                        return string_if_invalid % self.var
                    else:
                        return string_if_invalid
                else:
                    obj = string_if_invalid  # 99.9999... % of templates.
        ...
```

Fonctionnement fail-on-template-vars

La doc de django stipule : « If `string_if_invalid` contains a '%s', the format marker will be replaced with the name of the invalid variable. »

Idée diabolique :

- créer un objet personnalisé injecté en tant que `string_if_invalid`
- surcharger `__contains__` pour répondre `True` pour passer :

```
if "%s" in string_if_invalid:
```

- surcharger `__mod__` pour `self.fail()` le test :

```
    return string_if_invalid % self.var
```

Les deux problèmes

- Les tests échouent si une variable n'existe pas, même s'il y a un filtre `|default:`

```
<p>{{ user.phone|default:"Non renseigné" }}</p>
```

- Les `OneToOneField` dans les blocks `if` sont toujours évalués `True`

```
{% if user.supervisor %}
    Manager : {{ user.supervisor }}
{% endif %}
```

fail-on-template-vars : Variable

```python
class Variable:
    # {% if user.supervisor %}
    def _resolve_lookup(self, context):
        """Perform resolution of a real variable (i.e. not a literal) against the given context."""
        current = context
        try:  # catch-all for silent variable failures
            for bit in self.lookups:
                try:  # dictionary lookup
                    current = current[bit]
                except (TypeError, AttributeError, KeyError, ValueError, IndexError):
                    try:  # attribute lookup
                        ...
                        current = getattr(current, bit)
                    except (TypeError, AttributeError):
                        ...
                        try:  # list-index lookup
                            current = current[int(bit)]
                        except (IndexError, ValueError, KeyError, TypeError):
                            raise VariableDoesNotExist(
                                "Failed lookup for key [%s] in %r", (bit, current)
                            )
                # ... handle callables
        except Exception as e:
            template_name = getattr(context, "template_name", None) or "unknown"
            if getattr(e, "silent_variable_failure", False): # ObjectDoesNotExist and subclasses.
                current = context.template.engine.string_if_invalid  # Truthy when patched!!
```

FilterExpression

```
class FilterExpression:
    """Parse a variable token and its optional filters [...]"""
    # {{ user }}, {{ user | default:None }}
    def resolve(self, context, ignore_failures=False):
        ...
        try:
            obj = self.var.resolve(context)
        except VariableDoesNotExist:
            if ignore_failures:
                # FirstOfNode, ForNode, IfChangedNode, IfNode, RegroupNode.
                obj = None
            else:
                string_if_invalid = context.template.engine.string_if_invalid
                if string_if_invalid:
                    if "%s" in string_if_invalid:
                        return string_if_invalid % self.var
                    else:
                        return string_if_invalid
                else:
                    obj = string_if_invalid  # 99.9999... % of templates.
        ...
```

Patch 1 : |default and OneToOneField

```
@pytest.fixture(autouse=True, scope="session")
def _fail_for_invalid_template_variable_improved(_fail_for_invalid_template_variable):
    # Edge cases stuff omitted to fit on the slide.
    from django.conf import settings as dj_settings

    invalid_var_exception = dj_settings.TEMPLATES[0]["OPTIONS"]["string_if_invalid"]

    # Make InvalidVarException falsy to keep the behavior consistent for OneToOneField
    invalid_var_exception.__class__.__bool__ = lambda self: False

    # but adapt Django's template code to behave as if it was truthy in resolve
    # (except when the default filter is used)
    patchy.patch(
        base_template.FilterExpression.resolve,
        """\
        @@ -7,7 +7,8 @@
                         obj = None
                     else:
                         string_if_invalid = context.template.engine.string_if_invalid
        -                if string_if_invalid:
        +                from django.template.defaultfilters import default as default_filter
        +                if default_filter not in {func for func, _args in self.filters}:
                             if "%s" in string_if_invalid:
                                 return string_if_invalid % self.var
                             else:
        """,
    )
```

FAIL_INVALID_TEMPLATE_VARS

```
[tool.pytest.ini_options]
FAIL_INVALID_TEMPLATE_VARS = true
```

FAIL_INVALID_TEMPLATE_VARS

```toml
[tool.pytest.ini_options]
FAIL_INVALID_TEMPLATE_VARS = true
```

Des tests échouent ? Établir une baseline, avec :

```python
@pytest.mark.ignore_template_errors
def test_expected_failure():
    pass
```

Quizz

```
{% if somevar %}
    <p>More content</p>
{% endif %}
```

Quizz

```
{% if somevar %}
    <p>More content</p>
{% endif %}
```

« This behavior is slightly different for the `if`, `for` and `regroup` template tags. If an invalid variable is provided to one of these template tags, the variable will be interpreted as `None`. Filters are always applied to invalid variables within these template tags. »

fail-on-template-vars : FilterExpression

```
@register.tag("if")
def do_if(parser, token):
    # {% if ... %}
    bits = token.split_contents()[1:]
    condition = TemplateIfParser(parser, bits).parse()
    nodelist = parser.parse(("elif", "else", "endif"))
    conditions_nodelists = [(condition, nodelist)]
    token = parser.next_token()
    # {% elif ... %} (repeatable)
    ...
    # {% else %} (optional)
    ...
    # {% endif %}
    ...
    return IfNode(conditions_nodelists)

class TemplateIfParser(IfParser):
    def create_var(self, value):
        return TemplateLiteral(self.template_parser.compile_filter(value), value)

class TemplateLiteral(Literal):
    def __init__(self, value, text):
        self.value = value
        self.text = text  # for better error messages

    def display(self):
        return self.text

    def eval(self, context):
        return self.value.resolve(context, ignore_failures=True)
```

FilterExpression

```
class FilterExpression:
    """Parse a variable token and its optional filters [...]"""
    # {{ user }}, {{ user | default:None }}
    def resolve(self, context, ignore_failures=False):
        ...
        try:
            obj = self.var.resolve(context)
        except VariableDoesNotExist:
            if ignore_failures:
                # FirstOfNode, ForNode, IfChangedNode, IfNode, RegroupNode.
                obj = None
            else:
                string_if_invalid = context.template.engine.string_if_invalid
                if string_if_invalid:
                    if "%s" in string_if_invalid:
                        return string_if_invalid % self.var
                    else:
                        return string_if_invalid
                else:
                    obj = string_if_invalid  # 99.9999... % of templates.
        ...
```

Patch 2 : Remplacer `IGNORE_TEMPLATE_ERRORS`

```python
@pytest.fixture(autouse=True, scope="function")
def unknown_variable_template_error(monkeypatch, request):
    marker = request.keywords.get("ignore_unknown_variable_template_error", None)
    BASE_IGNORE_LIST = {"debug", "user"}
    strict = True
    if marker is None:
        ignore_list = BASE_IGNORE_LIST
    elif marker.args:
        ignore_list = BASE_IGNORE_LIST | set(marker.args)
    else:
        # Marker without list
        strict = False

    if strict:
        origin_resolve = base_template.FilterExpression.resolve
        # FirstOfNode, ForNode, IfChangedNode, IfNode, RegroupNode all force ignore_failures=True.
        def stricter_resolve(self, context, ignore_failures=False):
            if (
                self.is_var
                and self.var.lookups is not None
                and self.var.lookups[0] not in context
                and self.var.lookups[0] not in ignore_list
            ):
                ignore_failures = False
            return origin_resolve(self, context, ignore_failures)

        monkeypatch.setattr(base_template.FilterExpression, "resolve", stricter_resolve)
```

Sommaire

Le vénérable auto-submit

Recharger la page à chaque changement.
Démo

```
function submitFiltersForm() {
  $("#js-job-applications-filters-form").submit();
}
$("#js-job-applications-filters-form :input").change(submitFiltersForm);
$("duet-date-picker").on("duetChange", submitFiltersForm);
```

HTMX

Librairie JavaScript

- Fonctionnement typique :
  1. Définition des déclencheurs (`hx-trigger="click"`)
  2. Paramétrage du Fetch (url, méthode HTTP, headers, body)
  3. Le serveur génère la réponse (généralement un fragment `HTML`)
  4. Substitution d'une partie du DOM par la réponse
- Décrit par des attributs sur les éléments `HTML` (`hx-*`)

Démo `HTMX`

Recharger un fragment

Démo filtres candidatures

Template de la page

apply/job_application_list.html :

```
{% extends "layout/base.html" %}

{% block content %}
    <div class="s-section__row row">
        <div class="col-12">
            <h2 id="job-app-title">
                {{ job_apps|length }} résultat{{ job_apps|pluralize }}
            </h2>
        </div>
        <div class="col-12">
            <form hx-get="/apply/siae/list/"
                  hx-trigger="change"
                  hx-target="#job-app-results">
                {{ form.eligibility }}
            </form>
            {% include "apply/includes/job_app_results.html" %}
        </div>
    </div>
{% endblock content %}
```

Fragment pour HTMX

`apply/includes/job_app_results.html` :

```
<ul id="job-app-results">
    {% for job_app in job_apps %}
        <li>Candidature de {{ job_app.job_seeker.full_name }}</li>
    {% endfor %}
</ul>
```

La vue django

```python
def job_applications(request, *args, , **kwargs):
    # Access controls.
    context = {
        "job_apps": JobApplication.objects.filter(
            to_company=request.user.current_organization,
        )
    }
    template_name = (
        "apply/includes/job_app_results.html"
        if request.htmx
        else "apply/job_application_list.html"
    )
    return render(request, template_name, context)
```

Démo out-of-band swap

Démo actions préalables à l'embauche GEIQ

Template de base

apply/job_application_list.html :

```
{% extends "layout/base.html" %}

{% block content %}
    <div class="s-section__row row">
        <div class="col-12">
            {% include "apply/includes/job_app_title.html" %}
        </div>
        <div class="col-12">
            <form hx-get="/apply/siae/list/"
                  hx-trigger="change"
                  hx-target="#job-app-results">
                {{ form.eligibility }}
            </form>
            {% include "apply/includes/job_app_results.html" %}
        </div>
    </div>
{% endblock content %}
```

Fragments pour HTMX

apply/includes/job_app_title.html

```
<h2 id="job-app-title"{% if request.htmx %} hx-swap-oob="true"{% endif %}>
    {{ job_apps|length }} résultat{{ job_apps|pluralize }}
</h2>
```

apply/includes/job_app_results.html :

```
<ul id="job-app-results">
    {% for job_app in job_apps %}
        <li>Candidature de {{ job_app.job_seeker.full_name }}</li>
    {% endfor %}
</ul>
{% if request.htmx %}
    {% include "apply/includes/job_app_title.html" %}
{% endif %}
```

Danger des out-of-band

- Il est facile d'oublier de mettre à jour un fragment de la page

Sommaire

UPDATE_PAGE_WITH_HTMX

```python
def test_htmx_with_oob(self):
    self.client.force_login(self.user)
    response = self.client.get(self.URL, {"status": "NEW"})
    simulated_page = parse_response_to_soup(response)

    [new_status] = simulated_page.find_all(
        "input", attrs={"name": "status", "value": "NEW"},
    )
    del new_status["checked"]
    [ready_status] = simulated_page.find_all(
        "input", attrs={"name": "status", "value": "READY"},
    )
    ready_status["checked"] = ""
    response = self.client.get(
        self.URL, {"status": "READY"}, headers={"HX-Request": "true"}
    )
    update_page_with_htmx(
        simulated_page, f"form[hx-get='{self.URL}']", response
    )
    response = self.client.get(self.URL, {"status": "READY"})
    fresh_page = parse_response_to_soup(response)
    assertSoupEqual(simulated_page, fresh_page)
```

UPDATE_PAGE_WITH_HTMX 1/2

```python
def update_page_with_htmx(page, select_htmx_element, htmx_response):
    [htmx_element] = page.select(select_htmx_element)
    request_method = htmx_response.request["REQUEST_METHOD"]
    if request_method not in ("GET", "POST", "PUT", "DELETE", "PATCH"):
        raise ValueError(f"Unsupported method {request_method}")
    attribute = f"hx-{htmx_response.request['REQUEST_METHOD'].lower()}"
    if attribute not in htmx_element.attrs:
        raise ValueError(f"No {attribute} attribute on provided HTMX element")
    url = htmx_element[attribute]
    if url:
        # If url is "", it means that HTMX will have targeted the current URL
        # https://github.com/bigskysoftware/htmx/blob/v1.8.6/src/htmx.js#L2799-L2802
        # Let's not assert anything in that case, since we currently don't have that info in our test
        parsed_url = urlparse(url)
        assert htmx_response.request["PATH_INFO"] == parsed_url.path
    # We only support HTMX responses that do not try to swap the whole HTML body
    parsed_response = parse_response_to_soup(htmx_response, no_html_body=True)
```

UPDATE_PAGE_WITH_HTMX 2/2

```
out_of_band_swaps = [element.extract() for element in parsed_response.select("[hx-swap-oob]")]
for out_of_band_swap in out_of_band_swaps:
    oob_swap = out_of_band_swap["hx-swap-oob"]
    target_selector = None
    if oob_swap == "true":
        mode = "outerHTML"
    elif "," in oob_swap:
        mode, target_selector = oob_swap.split(",", maxsplit=1)
    else:
        mode = oob_swap
    del out_of_band_swap["hx-swap-oob"]
    if not target_selector:
        assert out_of_band_swap["id"], out_of_band_swap
        target_selector = f"#{out_of_band_swap['id']}"
    targets = page.select(target_selector)
    for target in targets:
        _handle_swap(page, target=target, new_elements=[out_of_band_swap], mode=mode)
_handle_swap(
    page,
    target=_get_hx_attribute(htmx_element, "hx-target"),
    new_elements=parsed_response.contents,
    mode=_get_hx_attribute(htmx_element, "hx-swap", default="innerHTML"),
)
```

_GET_HX_ATTRIBUTE

```python
def _get_hx_attribute(element, attribute, default=None):
    while (value := element.attrs.get(attribute)) is None:
        element = element.parent
        if element is None:
            if default is not None:
                return default
            raise ValueError(f"Attribute {attribute} not found on element or its parents")
    if attribute == "hx-target" and value == "this":
        return element
    return value
```

_HANDLE_SWAP

```python
def _handle_swap(page, *, target, new_elements, mode):
    if mode == "outerHTML":
        target_element = page.select_one(target) if isinstance(target, str) else target
        if not new_elements:
            # Empty response: remove the target completely
            target_element.decompose()
            return
        [first_element, *rest] = new_elements
        for rest_elt in reversed(rest):
            target_element.insert_after(rest_elt)
        target_element.replace_with(first_element)
        return
    raise NotImplementedError("Other kinds of swap not implemented, please do")
```

Merci de votre attention

Avez-vous des questions ?