django.js Documentation

Release 0.8.2.dev

Axel Haustant

August 18, 2015

Contents

Django.js provides tools for JavaScript development with Django.

Django.js is inspired from:

- Miguel Araujo's verbatim snippet.
- Dimitri Gnidash's django-js-utils.

Note: This is currently a work in progress (API wil not be stable before 1.0) so don't expect it to be perfect but please submit an issue for any bug you find or any feature you want.

CHAPTER 1

Compatibility

Django.js requires Python 2.6+ and Django 1.4.2+.

CHAPTER 2

Installation

You can install Django.js with pip:

```
$ pip install django.js
```

or with easy_install:

```
$ easy_install django.js
```

Add `djangojs` to your `settings.INSTALLED_APPS`.

Add `djangojs.urls` to your root `URL_CONF`:

```
urlpatterns = patterns('',
    ...
    url(r'^djangojs/', include('djangojs.urls')),
    ...
)
```

Note: Be sure to set the settings.ALLOWED_HOSTS properly (especially on Django 1.5+). If not, you will have HTTP 500 errors.

CHAPTER 3

Documentation

3.1 Template tags

3.1.1 Initialization

You can either:

- load the template tag lib into each template manually:

```
{% load js %}
```

- load the template tag lib by adding to your `views.py`:

```
from django.template import add_to_builtins

add_to_builtins('djangojs.templatetags.js')
```

If you want to use boolean parameters, Django.js provide the `djangojs.context_processors.booleans` to help. Simply add it to your `settings.CONTEXT_PROCESSORS`. If not, you should use the string versions: `param="True"`.

If `settings.DEBUG=True`, unminified versions are loaded (only for provided libraries, aka. Django.js, jQuery and jQuery Migrate).

3.1.2 Usage

django_js

A `{% django_js %}` tag is available to provide the Django JS module. After loading, you can use the Django module to resolve URLs and Translations:

```
{% django_js %}
<script>
    console.log(
        Django.url('my-view', {key: 'test'}),
        Django.file('test.json'),
        Django.context.STATIC_URL
    );
</script>
```

It supports the following keyword parameters (in this order if you want to omit the keyword):

Parameter	Default	Description
jquery	true	Load the jQuery library
i18n	true	Load the javascript i18n catalog
csrf	true	Patch jQuery.ajax() for Django CSRF
init	true	Initialize Django.js on load

You can disable all this features by simply providing arguments to the template tag:

```
{% django_js jquery=false i18n=false csrf=false %}
```

django_js_init

The `{% django_js_init %}` provide the necessary bootstrap for the Django.js without loading it. It allows you to use Django.js with an AMD loader or a javascript compressor. It supports the following keyword parameters (in this order if you want to omit the keyword):

Parameter	Default	Description
jquery	false	Load the jQuery library
i18n	true	Load the javascript i18n catalog
csrf	true	Patch jQuery.ajax() fot Django CSRF
init	true	Initialize Django.js on load

You can disable all this features by simply providing arguments to the template tag:

```
{% django_js_init jquery=true i18n=false csrf=false %}
```

If you want to use it with require.js or Django Pipeline, see RequireJS integration or *Django Pipeline*.

Internationalization

When the `{% django_js %}` template tag is included in a page, it automatically:

- loads the django javascript catalog for all supported apps
- **loads the django javascript i18n/l10n tools in the page:**
  - `gettext()`
  - `ngettext()`
  - `interpolate()`

You can disable this feature by setting the `i18n` keyword parameter to `false`.

Note: You can filter included apps by using either the settings whitelist `settings.JS_I18N` or the settings blacklist `settings.JS_I18N_EXCLUDE` or both. For more informations, see Settings.

jQuery Ajax CSRF

When the `django_js` template tag is ininitialized it automatically patch `jQuery.ajax()` to handle CSRF tokens on ajax request.

You can disable this feature by setting the `csrf` keyword parameter to `false`.

You can manually enable it later with:

```
Django.jquery_csrf();
```

verbatim

A `{% verbatim %}` tag is available to ease the JS templating. It escape a specific part. For example, you may want a subpart of your template to be rendered by Django :

```
<script type="text/x-handlebars" id="tpl-django-form">
    <form>
        {% verbatim %}
            {{#if id}}<h1>{{ id }}</h1>{{/if}}
        {% endverbatim %}
        {{ yourform.as_p }}
    </form>
</script>
```

Note: Starting from Django 1.5, use the included verbatim tag .

jquery_js

The `{% jquery_js %}` tag only load the jQuery library.

You can override the version either by passing the version as a parameter or setting the version with the `settings.JQUERY_VERSION` property. For more informations, see Settings.

You can optionnaly load the jQuery Migrate plugins for legacy support with jQuery 1.9.0+.

```
{% jquery_js %}
{% jquery_js "1.8.3" %}
{% jquery_js migrate=true %}
```

The `django_js` tag automatically load jQuery so no need to manually load it unless you set `jquery=false`.

javascript/js

The `javascript` and `js` tags are the same quick helper to include javascript files from `{{STATIC_URL}}`:

```
{% javascript "js/my.js" %}
{% js "js/my.js" %}
```

is equivalent to:

```
<script type="text/javascript" src="{% static "js/my.js" %}"></script>
```

Both tags take an options `type` parameter that specifies the content type of the `<script>` tag:

```
{% javascript "js/my.custom" type="text/custom" %}
```

yields:

```
<script type="text/custom" src="{% static "js/my.custom" %}"></script>
```

coffescript/coffee

The `coffeescript` and `coffee` tags are the same quick helper to include coffeescript files from `{{STATIC_URL}}`:

```
{% coffeescript "js/my.coffee" %}
{% coffee "js/my.coffee" %}
```

is equivalent to:

```
<script type="text/coffeescript" src="{% static "js/my.coffee" %}"></script>
```

css

The `css` tag is a quick helper to include css files from `{{STATIC_URL}}`:

```
{% css "css/my.css" %}
```

is equivalent to:

```
<link rel="stylesheet" type="text/css" href="{% static "css/my.css" %}" />
```

js_lib

The `js_lib` tag is a quick helper to include javascript files from `{{STATIC_URL}}js/libs`:

```
{% js_lib "my-lib.js" %}
```

is equivalent to:

```
<script type="text/javascript" src="{{STATIC_URL}}js/libs/my-lib.js"></script>
```

3.2 Django javascript module

3.2.1 Reverse URLs

The Django.js library expose reverse urls to javascript. You can call the `Django.url()` method with:

- an url name without arguments

```
Django.url('my-view');
```

- an url name and a variable number of arguments

```
Django.url('my-view', arg1, arg2);
```

- an url name and an array of arguments

```
Django.url('my-view', [arg1, arg2]);
```

- an url name and an object with named arguments

```
Django.url('my-view', {arg1: 'value1', arg2: 'value2'});
```

- an url name with one or more namespaces

```
Django.url('ns:my-view');
Django.url('ns:nested:my-view');
```

You can use anonymous forms (variable arguments and array) with named arguments in urls but you can't use object form with anonymous arguments.

You can also force unnamed URLs serialization with `settings.JS_URLS_UNNAMED`:

```
Django.url('path.to.my.view');
```

Note: You can filter included urls names and namespaces by using either the settings whitelists and blacklists: `settings.JS_URLS`, `settings.JS_URLS_EXCLUDE`, `settings.JS_URLS_NAMESPACES`, `settings.JS_URLS_NAMESPACES_EXCLUDE`.

For more informations, see Settings.

3.2.2 Static URLs

You can obtain a static file url with the `static` or `file` methods:

```
Django.static('my-data.json');
Django.file('my-data.json');
Django.static('another/data.pdf');
Django.file('another/data.pdf');
```

3.2.3 Context

Django.js wraps some Django values normally accessible in the template context:

- `Django.context.STATIC_URL`
- `Django.context.MEDIA_URL`
- `Django.context.LANGUAGES`
- `Django.context.LANGUAGE_CODE`
- `Django.context.LANGUAGE_NAME`
- `Django.context.LANGUAGE_NAME_LOCAL`
- `Django.context.LANGUAGE_BIDI`

In fact, any value contributed by a context processor and serializable will be accessible from `Django.context`.

3.2.4 User and permissions

Django.js allows you to check basic user attributes and permissions from client side. You can simply access the `Django.user` object or call the `Django.user.has_perm()` method:

```
console.log(Django.user.username);

if (Django.user.is_authenticated) {
    do_something();
}

if (Django.user.is_staff) {
```

```
    go_to_admin();
}

if (Django.user.is_superuser) {
    do_a_superuser_thing();
}

if (Django.user.has_perm('myapp.do_something')) {
    do_something();
}
```

Note: When using a custom user model with Django 1.5+, only the username and is_authenticated fields are significants. The other fields values will always be False or an empty tuple (for permissions), unless they exists on your custom model.

3.2.5 CSRF Tokens

Django.js provides some helpers for CSRF protection.

- return the value of the CSRF token

```
Django.csrf_token();
```

- return the hidden input element containing the CSRF token, like the `{% csrf_token %}` template tag

```
Django.csrf_element();
```

3.3 RequireJS integration

Django.js works with RequireJS but it requires some extras step to do it.

3.3.1 Preloading prerequisites

You should use the `django_js_init` template tag before loading your application with RequireJS.

```
{% load js %}
{% django_js_init %}
<script data-main="scripts/main" src="scripts/require.js"></script>
```

It works with django-require too:

```
{% load js require %}
{% django_js_init %}
{% require_module 'main' %}
```

See *django_js_init*.

3.3.2 shim configuration

You should add an extra shim configuration for Django.js:

```
require.config({
    paths: {
        django: 'djangojs/django'
    },

    shim: {
        "django": {
            "deps": ["jquery"],
            "exports": "Django"
        }
    }
});
```

3.4 Javascript test tools

Django.js provide tools for easy javascript testing.

3.4.1 Views

Django.js provides base views for javascript testing. Instead of writing a full view each time you need a Jasmine or a QUnit test view, simply use the provided `JasmineView` and `QUnitView` and add them to your test_urls.py:

```
from django.conf.urls import patterns, url, include

from djangojs.views import JasmineView, QUnitView

urlpatterns = patterns('',
    url(r'^jasmine$', JasmineView.as_view(js_files='js/specs/*.specs.js'), name='my_jasmine_view'),
    url(r'^qunit$', QUnitView.as_view(js_files='js/tests/*.tests.js'), name='my_qunit_view'),
)
```

Both view have a js_files attribute which can be a string or and array of strings. Each string can be a static js file path to include or a glob pattern:

```
from djangojs.views import JasmineView

class MyJasmineView(JasmineView):
    js_files = (
        'js/lib/my-lib.js',
        'js/test/*.specs.js',
        'js/other/specs.*.js',
    )
```

Note: Files order matters and will be preserved.

jQuery can automatically be included into the view by setting the `jquery` attribute to `True`:

```
from djangojs.views import JasmineView

class MyJasmineView(JasmineView):
    jquery = True
    js_files = 'js/test/*.specs.js'
```

Django.js can automatically be included into the view by setting the `django_js` attribute to `True`:

```
from djangojs.views import JasmineView

class MyJasmineView(JasmineView):
    django_js = True
    js_files = 'js/test/*.specs.js'
```

These views extends the Django `TemplateView` so you can add extra context entries and you can customize the template by extending them.

```
from djangojs.views import QUnitView

class MyQUnitView(QUnitView):
    js_files = 'js/test/*.test.js'
    template_name = 'my-qunit-runner.html'

    def get_context_data(self, **kwargs):
        context = super(MyQUnitView, self).get_context_data(**kwargs)
        context['form'] = TestForm()
        return context
```

Two extensible test runner templates are provided:

- `djangojs/jasmine-runner.html` for jasmine tests
- `djangojs/qunit-runner.html` for QUnit tests

Both provides a `js_init` block, a `js_content` block and a `body_content` block.

```
{% extends "djangojs/qunit-runner.html" %}

{% block js_init %}
    {{ block.super }}
    {% js "js/init.js" %}
{% endblock %}

{% block js_content %}
    {% load js %}
    {% js "js/tests/my.tests.js" %}
{% endblock %}

{% block body_content %}
  <form id="test-form" action="{% url test_form %}" method="POST" style="display: none;">
    {{csrf_token}}
    {{form}}
  </form>
{% endblock %}
```

You can inspect django.js own test suites on github.

If you just need the Django.js comptible runners, you can include the following templates (depending on your framework):

- **QUnit:**
  - `djangojs/qunit-runner-head.html`
  - `djangojs/qunit-runner-body.html`
- **Jasmine:**
  - `djangojs/jasmine-runner-head.html`
  - `djangojs/jasmine-runner-body.html`

3.4.2 Test cases

A Phantom.js test runner parsing TAP is provided in 3 flavours:

- `JsTestCase` that runs javascript tests against Django liveserver TestCase.
- `JsFileTestCase` that runs javascript tests against a static html file
- `JsTemplateTestCase` that runs javascript tests against a rendered html file (but without liveserver running)

Note: Whatever TestCase you choose, it should output TAP. If you don't have complex and specific needs, you just have to use the provided template and extends them if needed.

Jasmine/QUnit support are provided with `JasmineSuite` and `QUnitSuite` mixins.

To use it with the previously defined views, just define either `url_name` or `filename` attribute:

```
from djangojs.runners import JsTestCase, JsFileTestCase, JsTemplateTestCase
from djangojs.runners import JasmineSuite, QUnitSuite


class JasminTests(JasmineSuite, JsTestCase):
    urls = 'myapp.test_urls'
    title = 'My Jasmine suite'
    url_name = 'my_url_name'


class QUnitTests(QunitSuite, JsFileTestCase):
    filename = '/tmp/my-runner.html'


class JasminTests(JasmineSuite, JsTemplateTestCase):
    template_name = 'my/template.html'
    js_files = 'js/test/other/*.js'
```

The verbosity is automatically adjusted with the `-v/--verbosity` parameter from the `manage.py test` command line.

Warning: Phantom.js is required to use this feature and should be on your `$PATH`.

3.5 Integration with other Django apps

3.5.1 Django Absolute

Django Absolute contribute with the following context variables:

- `ABSOLUTE_ROOT`
- `ABSOLUTE_ROOT_URL`
- `SITE_ROOT`
- `SITE_ROOT_URL`

They will be available into `Django.context` javascript object (nothing new, this the default behavior). But, two more methods will be available:

- `Django.absolute()` to reverse an absolute URL based on request

- `Django.site()` to reverse an absolute URL based on Django site

If you try to call these methods without django-bsolute installed, a `DjangoJsError` will be thrown.

3.5.2 Django Pipeline

If you want to compress Django.js with Django Pipeline, you should change the way you load django.js.

First add jQuery and Django.js to your pipelines in you `settings.py`:

```
PIPELINE_JS = {
    'base': {
        'source_filenames': (
            '...',
            'js/libs/jquery-1.9.1.js',
            'js/djangojs/django.js',
            '...',
        ),
        'output_filename': 'js/base.min.js',
    },
}
```

Instead of using the `django_js` template tag:

```
{% load js %}
{% django_js %}
```

you should use the *django_js_init* and include your compressed bundle:

```
{% load js compressed %}
{% django_js_init %}
{% compressed_js "base" %}
```

3.6 Management Commands

Django.js provide a management command to simplify some common JavaScript tasks:

```
$ python manage.py js -h
usage: manage.py js [-h] [-v {0,1,2,3}] [--settings SETTINGS]
                    [--pythonpath PYTHONPATH] [--traceback]
                    {bower,launcher,localize} ...

Handle javascript operations

optional arguments:
  -h, --help            show this help message and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Print traceback on exception
```

```
subcommands:
  JavaScript command to execute

  {bower,launcher,localize}
    bower               Generate a .bowerrc file
    launcher            Get a PhantomJS launcher path
    localize            Generate PO file from js files
```

3.6.1 `localize`

The `localize` command generates a `.po` file for your javascript files. It allows you to localize your templates with custom patterns.

Custom patterns are specified in *settings.JS_I18N_PATTERNS*.

Let says you use Handlebars as client-side template engine, all your templates are `.hbs` files in your app `static/templates` directory and you registered a `trans` helper to handle localization:

```
Handlebars.registerHelper('trans', function(opt) {
    return gettext(opt.fn(this));
});
```

So, in your handlebars templates, you will have some localizable strings like:

```
{{#trans}}my translatable label{{/trans}}
```

You can add this to your `settings.py` file to extract localizable strings from them:

```
JS_I18N_PATTERNS = (
    ('hbs', 'static/templates', r'{{#trans}}(.*?){{/trans}}'),
)
```

Running the localize command:

```
$ python manage.py js localize -l fr
```

will extract all localizable strings from your `.js` files as usual and add those in your `.hbs` files.

```
$ python manage.py js localize -h
usage: manage.py js localize [-h] [--locale LOCALE] [--all]
                             [--extension EXTENSIONS] [--symlinks]
                             [--ignore PATTERN] [--no-default-ignore]
                             [--no-wrap] [--no-location] [--no-obsolete]
                             [app [app ...]]

Generate PO file from js files

positional arguments:
  app                   Applications to localize

optional arguments:
  -h, --help            show this help message and exit
  --locale LOCALE, -l LOCALE
                        Creates or updates the message files for the given
                        locale (e.g. pt_BR).
  --all, -a             Updates the message files for all existing locales.
  --extension EXTENSIONS, -e EXTENSIONS
                        The file extension(s) to examine (default: "js").
```

```
                        Separate multiple extensions with commas, or use -e
                        multiple times.
  --symlinks, -s        Follows symlinks to directories when examining source
                        code and templates for translation strings.
  --ignore PATTERN, -i PATTERN
                        Ignore files or directories matching this glob-style
                        pattern. Use multiple times to ignore more.
  --no-default-ignore   Don't ignore the common glob-style patterns 'CVS',
                        '.*' and '*~'.
  --no-wrap             Don't break long message lines into several lines
  --no-location         Don't write '#: filename:line' lines
  --no-obsolete         Remove obsolete message strings
```

3.6.2 bower

The `bower` command generates a `.bowerrc` file into the current directory specifying the target directory for Bower downloads.

```
$ python manage.py js bower -h
usage: manage.py js bower [-h] [-f] target

Generate a .bowerrc file

positional arguments:
  target       The target directory for bower downloads

optional arguments:
  -h, --help   show this help message and exit
  -f, --force  Overwrite the file if exists
```

exemple:

```
$ python manage.py js bower myproject/static/bower
Created .bowerrc file into the current directory
$ cat .bowerrc
{"directory": "./myproject/static/bower/"}
```

3.6.3 launcher

The `launcher` command returns the full path to a Django.js PhantomJS runner (usefull if you need to execute it manually).

```
$ python manage.py js launcher -h
usage: manage.py js launcher [-h] name

Get a PhantomJS launcher path

positional arguments:
  name        Runner name

optional arguments:
  -h, --help  show this help message and exit
```

exemple:

```
$ python manage.py js launcher jasmine
/var/lib/python2.7/site-packages/django.js/djangojs/phantomjs/jasmine-runner.js
```

3.7 Settings

You can tune Django.js behaviour using settings. Django.js provide the following optionnal settings:

3.7.1 Libraries versions

You can specify some libraries versions used by Django.js.

JQUERY_VERSION

Specify the jQuery version to use. If not specififed, default to last version.

Django.js provide the following versions:

- 2.0.3
- 2.0.2
- 2.0.1
- 2.0.0
- 1.10.2
- 1.10.1
- 1.9.1
- 1.9.0
- 1.8.3

3.7.2 URLs handling

Theses settings allow you to customize or disable Django.js URLs handling.

JS_URLS_ENABLED

Default: `True`

You can disable Django.js URLs handling by setting it to `False`

JS_URLS

Default: `None`

Serialized URLs names whitelist. If this setting is specified, only named URLs listed in will be serialized.

JS_URLS_EXCLUDE

Default: `None`

Serialized URLs names blacklist. It this setting is specified, named URLs listed in will not be serialized.

JS_URLS_NAMESPACES

Default: `None`

Serialized namespaces whitelist. If this setting is specified, only URLs from namespaces listed in will be serialized.

JS_URLS_NAMESPACES_EXCLUDE

Default: `None`

Serialized namespaces blacklist. If this setting is specified, URLs from namespaces listed in will not be serialized.

JS_URLS_UNNAMED

Default: `False`

Serialize unnamed URLs. If this setting is set to `True`, unnamed URLs will be serialized (only for function based views).

3.7.3 Context handling

Theses settings allow you to customize or disable Django.js context handling.

JS_CONTEXT_ENABLED

default: `True`

You can disable Django.js context handling by setting it to `False`

JS_CONTEXT

default: `None`

Serialized context variables names whitelist. If this setting is specified, only context variables listed in will be serialized.

Note: `LANGUAGE_NAME` and `LANGUAGE_NAME_LOCAL` requires `LANGUAGE_CODE` to be also included.

JS_CONTEXT_EXCLUDE

Default: `None`

Serialized context variables names blacklist. If this setting is specified, context variables names listed in will not be serialized.

Note: Excluding `LANGUAGE_CODE` also exclude `LANGUAGE_NAME` and `LANGUAGE_NAME_LOCAL`.

JS_CONTEXT_PROCESSOR

Default: `djangojs.utils.ContextSerializer`

Change this value if you want to specify a custom context serializer class. The custom class must inherits from `ContextSerializer`

3.7.4 User handling

JS_USER_ENABLED

default: `True`

You can disable Django.js user handling by setting it to `False`

3.7.5 Localization and internationalization

JS_I18N_APPS

Default: `None`

Serialized translations whitelist. If specified, only apps listed in will appear in the javascript translation catalog.

JS_I18N_APPS_EXCLUDE

Default: `None`

Serialized translations blacklist. If specified, apps listed in will not appear in the javascript translation catalog.

JS_I18N_PATTERNS

Default: `tuple()`

Custom patterns for localization using the *localize management command*. Each entry should be a tuple `(extension, dirname, pattern)` where:

extension is an file extension to match

dirname is the application relative path to search into

pattern is a expressions to extract localizable strings (can be a list of regular expressions).

Exemple:

```
JS_I18N_PATTERNS = (
    ('hbs', 'static/templates', r'{{#trans}}(.*?){{/trans}}'),
)
```

3.7.6 Cache management

JS_CACHE_DURATION

Default: `24 * 60` (24 hours)

Django.js urls and context cache duration in minutes.

Usage exemple

You could have, in your `settings.py`:

```
# Exclude my secrets pages from serialized URLs
JS_URLS_EXCLUDE = (
    'my_secret_page',
    'another_secret_page',
)
# Only include admin namespace
JS_URLS_NAMESPACES = (
    'admin',
)
# Only include my apps' translations
JS_I18N_APPS = ('myapp', 'myapp.other')

# Disable user serialization
JS_USER_ENABLED = False

# Custom Context serializer
JS_CONTEXT_PROCESSOR = 'my.custom.ContextProcessor'
```

3.8 API

3.8.1 `djangojs` – Main package

Django.js provide better integration of javascript into Django.

`djangojs.`**JQUERY_DEFAULT_VERSION = '2.0.3'**
Packaged jQuery version

3.8.2 `djangojs.context_serializer` – Context serialization handling

class `djangojs.context_serializer.`**ContextSerializer**(*request*)
Bases: `object`

Serialize context and user from requests.

Inherits from this class and set your *settings.JS_CONTEXT_PROCESSOR* to customize the serialization.

To add a custom variable serialization handler, add a method named `process_VARNAME(self, value, data)`.

as_dict()
Serialize the context as a dictionnary from a given request.

as_json()
Serialize the context as JSON.

handle_user(*data*)
Insert user informations in data

Override it to add extra user attributes.

process_LANGUAGES(*languages*, *data*)
Serialize LANGUAGES as a localized dictionnary.

process_LANGUAGE_CODE(*language_code*, *data*)
Fix language code when set to non included default *en* and add the extra variables `LANGUAGE_NAME` and `LANGUAGE_NAME_LOCAL`.

3.8.3 `djangojs.views` – Javascript views helpers

This module provide helper views for javascript.

class `djangojs.views.`**JsInitView**(***kwargs*)
Bases: `djangojs.views.UserCacheMixin`, `django.views.generic.base.TemplateView`

Render a javascript file containing the URLs mapping and the context as JSON.

class `djangojs.views.`**JsonView**(***kwargs*)
Bases: `django.views.generic.base.View`

A views that render JSON.

class `djangojs.views.`**UrlsJsonView**(***kwargs*)
Bases: `djangojs.views.CacheMixin`, *djangojs.views.JsonView*

Render the URLs as a JSON object.

class `djangojs.views.`**ContextJsonView**(***kwargs*)
Bases: `djangojs.views.UserCacheMixin`, *djangojs.views.JsonView*

Render the context as a JSON object.

class `djangojs.views.`**JsTestView**(***kwargs*)
Bases: `django.views.generic.base.TemplateView`

Base class for JS tests views

django_js = False
Includes or not Django.js in the test view

django_js_init = True
Initialize or not Django.js in the test view (only if included)

jquery = False
Includes or not jQuery in the test view.

js_files = None
A path or a list of path to javascript files to include into the view.

- Supports glob patterns.
- Order is kept for rendering.

class `djangojs.views.`**JasmineView**(***kwargs*)
Bases: *djangojs.views.JsTestView*

Render a Jasmine test runner.

class `djangojs.views.`**`QUnitView`**(***kwargs*)
Bases: *djangojs.views.JsTestView*

Render a QUnit test runner

`theme` = u'qunit'
QUnit runner theme.

Should be one of: qunit, gabe, ninja, nv

3.8.4 `djangojs.runners` – Javascript unittest runners

This module provide Javascript test runners for Django unittest.

class `djangojs.runners.`**`PhantomJsRunner`**
Bases: `object`

Test helper to run JS tests with PhantomJS

`execute`(*command*)
Execute a subprocess yielding output lines

`phantomjs`(**args*, ***kwargs*)
Execute PhantomJS by giving `args` as command line arguments.

If test are run in verbose mode (`-v/--verbosity` = 2), it output:

- the title as header (with separators before and after)
- modules and test names
- assertions results (with `django.utils.termcolors` support)

In case of error, a JsTestException is raised to give details about javascript errors.

`phantomjs_runner` = None
mandatory path to the PhantomJS javascript runner

`run_suite`()
Run a phantomjs test suite.

•`phantomjs_runner` is mandatory.

•Either `url` or `url_name` needs to be defined.

`timeout` = 3
PhantomJS execution timeout in seconds

`title` = u'PhantomJS test suite'
an optionnal title for verbose console output

`url` = None
an optionnal absolute URL to the test runner page

class `djangojs.runners.`**`JsTestCase`**(*methodName='runTest'*)
Bases: *djangojs.runners.PhantomJsRunner*, `django.test.testcases.LiveServerTestCase`

A PhantomJS suite that run against the Django LiveServerTestCase

`url_args` = None
an optionnal arguments array to pass to the `reverse()` function

`url_kwargs` = None
an optionnal keyword arguments dictionnary to pass to the `reverse()` function

url_name = None
a mandatory named URL that point to the test runner page

class djangojs.runners.**JsFileTestCase**(*methodName='runTest'*)
Bases: *djangojs.runners.PhantomJsRunner*, unittest.case.TestCase

A PhantomJS suite that run against a local html file

filename = None
absolute path to the test runner page

class djangojs.runners.**JsTemplateTestCase**(*methodName='runTest'*)
Bases: *djangojs.runners.JsFileTestCase*

A PhantomJS suite that run against a rendered html file but without server.

Note: Template is rendered using a modified static storage that give `file://` scheme URLs. To benefits from it, you have to use either the `static` template tag or one the djangojs template tags.

Warning: Template is not rendered within a request/response dialog. You can't access the request object and everything that depends on the server.

jquery = False
Includes or not jQuery in the test view. Template has to handle the `use_jquery` property.

js_files = None
A path or a list of path to javascript files to include into the view.

- Supports glob patterns.
- Order is kept for rendering.

template_name = None
absolute path to the test runner page

exception djangojs.runners.**JsTestException**(*message*, *failures=None*)
Bases: exceptions.Exception

An exception raised by Javascript tests.

It display javascript errors into the exception message.

class djangojs.runners.**JasmineSuite**
Bases: object

A mixin that runs a jasmine test suite with PhantomJs.

class djangojs.runners.**QUnitSuite**
Bases: object

A mixin that runs a QUnit test suite with PhantomJs.

class djangojs.runners.**AbsoluteFileStorage**(*location=None*, *base_url=None*, *file_permissions_mode=None*, *directory_permissions_mode=None*)
Bases: django.core.files.storage.FileSystemStorage

A storage that give the absolute file scheme URL as URL.

3.8.5 djangojs.urls_serializer – URLs serialization handling

djangojs.urls_serializer.**urls_as_dict**()
Get the URLs mapping as a dictionnary

djangojs.urls_serializer.**urls_as_json**()
Get the URLs mapping as JSON

3.8.6 djangojs.utils – Miscellaneous helpers

This modules holds every helpers that does not fit in any standard django modules.

djangojs.utils.**class_from_string**(*name*)
Get a python class object from its name

class djangojs.utils.**LazyJsonEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Bases: django.core.serializers.json.DjangoJSONEncoder

A JSON encoder handling promises (aka. Django lazy objects).

See: https://docs.djangoproject.com/en/dev/topics/serialization/#id2

class djangojs.utils.**StorageGlobber**
Bases: object

Retrieve file list from static file storages.

classmethod glob(*files=None*)
Glob a pattern or a list of pattern static storage relative(s).

3.8.7 djangojs.tap – Tap format parser

This module provide test runners for JS in Django.

class djangojs.tap.**TapParser**(*yield_class=<class 'djangojs.tap.TapTest'>*, *debug=False*)
Bases: object

A TAP parser class reading from iterable TAP lines.

3.8.8 djangojs.templatetags.js – Javascript template tags

Provide template tags to help with Javascript/Django integration.

class djangojs.templatetags.js.**VerbatimNode**(*text_and_nodes*)
Bases: django.template.base.Node

Wrap {% verbatim %} and {% endverbatim %} around a block of javascript template and this will try its best to output the contents with no changes.

```
{% verbatim %}
    {% trans "Your name is" %} {{first}} {{last}}
{% endverbatim %}
```

djangojs.templatetags.js.**coffee**(*filename*)
A simple shortcut to render a script tag to a static coffeescript file

djangojs.templatetags.js.**coffeescript**(*filename*)
A simple shortcut to render a `script` tag to a static coffeescript file

djangojs.templatetags.js.**css**(*filename*)
A simple shortcut to render a `link` tag to a static CSS file

djangojs.templatetags.js.**django_js**(*context*, *jquery=True*, *i18n=True*, *csrf=True*, *init=True*)
Include Django.js javascript library in the page

djangojs.templatetags.js.**django_js_init**(*context*, *jquery=False*, *i18n=True*, *csrf=True*, *init=True*)
Include Django.js javascript library initialization in the page

djangojs.templatetags.js.**javascript**(*filename*, *type=u'text/javascript'*)
A simple shortcut to render a `script` tag to a static javascript file

djangojs.templatetags.js.**jquery_js**(*version=None*, *migrate=False*)
A shortcut to render a `script` tag for the packaged jQuery

djangojs.templatetags.js.**js**(*filename*, *type=u'text/javascript'*)
A simple shortcut to render a `script` tag to a static javascript file

djangojs.templatetags.js.**verbatim**(*parser*, *token*)
Renders verbatim tags

djangojs.templatetags.js.**verbatim_tags**(*parser*, *token*, *endtagname*)
Javascript templates (jquery, handlebars.js, mustache.js) use constructs like:

```
{{if condition}} print something{{/if}}
```

This, of course, completely screws up Django templates, because Django thinks {{ and }} means something.

The following code preserves {{ }} tokens.

This version of verbatim template tag allows you to use tags like url {% url name %}. {% trans "foo" %} or {% csrf_token %} within.

Inspired by:

- Miguel Araujo: https://gist.github.com/893408

3.8.9 `djangojs.context_processors` – Context processors

djangojs.context_processors.**booleans**(*request*)
Allow to use booleans in templates.

See: http://stackoverflow.com/questions/4557114/django-custom-template-tag-which-accepts-a-boolean-parameter

3.8.10 `djangojs.contrib` – Contributed compatibility modules

This package is dedicated to contributed compatibility modules.

These modules provide mixins and already packed *`ContextSerializers`* to use Django applications incompatible with the defaut Django.js behavior.

`djangojs.contrib.social_auth` – django_social_auth support

This module provide support for django_social_auth.

class `djangojs.contrib.social_auth.`**`SocialAuthContextMixin`**
Bases: `object`

Handle django_social_auth context specifics

`process_social_auth`(*social_auth*, *data*)
Just force social_auth's LazyDict to be converted to a dict for the JSON serialization to work properly.

class `djangojs.contrib.social_auth.`**`SocialAuthContextSerializer`**(*request*)
Bases: *djangojs.contrib.social_auth.SocialAuthContextMixin*, *djangojs.context_serializer.ContextSerializer*

Already packed django_social_auth ContextSerializer

3.9 Contributing

Django.js is open-source and very open to contributions.

3.9.1 Submitting issues

Issues are contributions in a way so don't hesitate to submit reports on the official bugtracker.

Provide as much informations as possible to specify the issues:

- the Django.js version used
- a stacktrace
- installed applications list
- ...

3.9.2 Submitting patches (bugfix, features, ...)

If you want to contribute some code:

1. fork the official Django.js repository
2. create a branch with an explicit name (like `my-new-feature` or `issue-XX`)
3. do your work in it
4. rebase it on the master branch from the official repository (cleanup your history by performing an interactive rebase)
5. submit your pull-request

There are some rules to follow:

- your contribution should be documented (if needed)
- your contribution should be tested and the test suite should pass successfully
- your code should be mostly PEP8 compatible with a 120 characters line length
- your contribution should support both Python 2 and 3 (use `tox` to test)

You need to install some dependencies to hack on Django.js:

```
$ pip install -r requirements/develop.pip
```

A Makefile is provided to simplify the common tasks:

```
$ make
Makefile for Django.js

Usage:
   make serve            Run the test server
   make test             Run the test suite
   make coverage         Run a caoverage report from the test suite
   make pep8             Run the PEP8 report
   make pylint           Run the pylint report
   make doc              Generate the documentation
   make minify           Minify all JS files with yuglify
   make dist             Generate a distributable package
   make clean            Remove all temporary and generated artifacts
```

To ensure everything is fine before submission, use `tox`. It will run the test suite on all the supported Python version and ensure the documentation is generating.

```
$ pip install tox
$ tox
```

You also need to ensure your code is PEP8 compliant (following the project rules: see `pep8.rc` file):

```
$ make pep8
```

Don't forget client-side code and tests.

You can run the javascript test suite in the browser (http://localhost:8000). Javascript tests are run in the test suite too, but it runs on the minified version of the javascript libary.

You can use the Makefile `minify` task that minify the javascript:

```
$ make minify test
```

Note: minification use `yuglify` so you need to install it before: `npm install -g yuglify`

3.10 Changelog

3.10.1 Current

- Nothing yet

3.10.2 0.8.1 (2013-10-19)

- Fixed management command with Django < 1.5 (fix issue #23 thanks to Wasil Sergejczyk)
- Fixed Django CMS handling (fix issue #25 thanks to Wasil Sergejczyk)
- Cache Django.js views and added `settings.JS_CACHE_DURATION`
- Allow customizable Django.js initialization

- Allow manual reload of context and URLs
- Published Django.js on bower (thanks to Wasil Sergejczyk for the initial bower.json file)
- Do not automatically translate languages name in context

3.10.3 0.8.0 (2013-07-14)

- **Allow features to be disabled with:**
  - `settings.JS_URLS_ENABLED`
  - `settings.JS_USER_ENABLED`
  - `settings.JS_CONTEXT_ENABLED`
- Added context black and white lists (`settings.JS_CONTEXT` and `settings.JS_CONTEXT_EXCLUDE`)
- Allow context serialization customization by inheritance with `settings.JS_CONTEXT_PROCESSOR`
- Do not fail on import when parsing URLs (Fix issue #7 thanks to Wasil Sergejczyk)
- Treat starred non-capturing groups and starred characters as optionnals (Fix issue #22)
- Upgraded to jQuery 2.0.3 (and added 1.10.2)
- Upgraded to QUnit 1.12.0
- Added `js` management command.
- Extracted URLs handling and context handling into their own modules
- First contrib module: `social_auth` (thanks to Olivier Cortès)

3.10.4 0.7.6 (2013-06-07)

- Reintroduce Python 2.6 support (thanks to Andy Freeland)
- Fix issue #20

3.10.5 0.7.5 (2013-06-01)

- Handle Django 1.5+ custom user model
- Upgraded to jQuery 2.0.2 and jQuery Migrate 1.2.1

3.10.6 0.7.4 (2013-05-11)

- Preserve declaration order in StorageGlobber.glob() (Fix issue #17)
- Fixes on localization on handling

3.10.7 0.7.3 (2013-04-30)

- Upgraded to jQuery 2.0.0
- Package both minified and unminified versions.
- Load minified versions (Django.js, jQuery and jQuery Migrate) when DEBUG=False

3.10.8 0.7.2 (2013-04-30)

- Fix issue #16
- Declare package as Python 3 compatible on PyPI

3.10.9 0.7.1 (2013-04-25)

- Optionnaly include jQuery with `{% django_js_init %}`.

3.10.10 0.7.0 (2013-04-25)

- Added RequireJS/AMD helpers and documentation
- Added Django Pipeline integration helpers and documentation
- Support unnamed URLs resolution.
- Support custom content types to be passed into the js/javascript script tag (thanks to Travis Jensen)
- Added `coffee` and `coffescript` template tags
- Python 3 compatibility

3.10.11 0.6.5 (2013-03-13)

- Make JsonView reusable
- Unescape regex characters in URLs
- Fix handling of 0 as parameter for Javasript reverse URLs

3.10.12 0.6.4 (2013-03-10)

- Support namespaces without app_name set.

3.10.13 0.6.3 (2013-03-08)

- Fix CSRF misspelling (thanks to Andy Freeland)
- Added some client side CSRF helpers (thanks to Andy Freeland)
- Upgrade to jQuery 1.9.1 and jQuery Migrate 1.1.1
- Do not clutter url parameters in `js`, `javascript` and `js_lib` template tags.

3.10.14 0.6.2 (2013-02-18)

- Compatible with Django 1.5

3.10.15 0.6.1 (2013-02-11)

- Added `static` method (even if it's a unused reserved keyword)

3.10.16 0.6 (2013-02-09)

- Added basic user attributes access
- Added permissions support
- Added `booleans` context processor
- Added jQuery 1.9.0 and jQuery Migrate 1.0.0
- Upgraded QUnit to 1.11.0
- Added QUnit theme support
- Allow to specify jQuery version (1.8.3 and 1.9.0 are bundled)

3.10.17 0.5 (2012-12-17)

- Added namespaced URLs support
- Upgraded to Jasmine 1.3.1
- **Refactor testing tools:**
  - Rename `test/js` into `js/test` and reorganize test resources
  - Renamed `runner_url*` into `url*` on `JsTestCase`
  - Handle `url_args` and `url_kwargs` on `JsTestCase`
  - Renamed `JasmineMixin` into `JasmineSuite`
  - Renamed `QUnitMixin` into `QUnitSuite`
  - Extracted runners initialization into includable templates
- Added `JsFileTestCase` to run tests from a static html file without live server
- Added `JsTemplateTestCase` to run tests from a rendered template file without live server
- **Added some settings to filter scope:**
  - Serialized named URLs whitelist: `settings.JS_URLS`
  - Serialized named URLs blacklist: `settings.JS_URLS_EXCLUDE`
  - Serialized namespaces whitelist: `settings.JS_URLS_NAMESPACES`
  - Serialized namespaces blacklist: `settings.JS_URLS_NAMESPACES_EXCLUDE`
  - Serialized translations whitelist: `settings.JS_I18N_APPS`
  - Serialized translations blacklist: `settings.JS_I18N_APPS_EXCLUDE`
- Expose PhantomJS timeout with `PhantomJsRunner.timeout` attribute

3.10.18 0.4 (2012-12-04)

- Upgraded to jQuery 1.8.3
- Upgraded to Jasmine 1.3.0
- Synchronous URLs and context fetch.
- Use `django.utils.termcolors`

- **Class based javascript testing tools:**
  - Factorize `JsTestCase` common behaviour
  - Removed `JsTestCase.run_jasmine()` and added `JasmineMixin`
  - Removed `JsTestCase.run_qunit()` and added `QUnitMixin`
  - Extract `TapParser` into `djangojs.tap`
- Only one Django.js test suite
- Each framework is tested against its own test suite
- Make jQuery support optionnal into `JsTestCase`
- Improved JsTestCase output
- Drop Python 2.6 support
- Added API documentation

3.10.19 0.3.2 (2012-11-10)

- Optionnal support for Django Absolute

3.10.20 0.3.1 (2012-11-03)

- Added JsTestView.django_js to optionnaly include django.js
- Added js_init block to runners to templates.

3.10.21 0.3 (2012-11-02)

- Improved `ready` event handling
- Removed runners from `urls.py`
- Added documentation
- Added `ContextJsonView` and `Django.context` fetched from json.
- Improved error handling
- Added `DjangoJsError` custom error type

3.10.22 0.2 (2012-10-23)

- Refactor template tag initialization
- Provides Jasmine and QUnit test views with test discovery (globbing)
- Provides Jasmine and QUnit test cases
- Added `Django.file()`
- Added `{% javascript %}`, `{% js %}` and `{% css %}` template tags

3.10.23 0.1.3 (2012-10-02)

- First public release
- Provides django.js with `url()` method and constants
- Provides `{% verbatim %}` template tag
- Patch `jQuery.ajax()` to handle CSRF tokens
- Loads the django javascript catalog for all apps supporting it
- Loads the django javascript i18n/l10n tools in the page

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

d

A

B

C

D

E

F

G

H

J

L

P

Q

R

S

T

U

V