django.js Documentation

Release 0.7.0

Axel Haustant

April 25, 2013

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

ONE

COMPATIBILITY

Django.js requires Python 2.7+ and Django 1.4.2+.

CHAPTER

TWO

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')),
    ...
)
```

CHAPTER

THREE

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"`.

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() fot Django CSRF

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
i18n	true	Load the javascript i18n catalog
csrf	true	Patch jQuery.ajax() fot Django CSRF

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

```
{% django_js_init 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();
}
```

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',
    )
```

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.min.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 Settings

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

3.6.1 JQUERY_VERSION

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

Django.js provide the following versions:

- 1.8.3
- 1.9.0
- 1.9.1

3.6.2 JS_URLS

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

- Default value: `None`
- Expected: a list of URLs names to include only

3.6.3 JS_URLS_EXCLUDE

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

- Default value: `None`
- Expected: a list of URLs names to exclude

3.6.4 JS_URLS_NAMESPACES

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

- Default value: `None`
- Expected: a list of URL namespaces to include only

3.6.5 JS_URLS_NAMESPACES_EXCLUDE

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

- Default value: `None`
- Expected: a list of URL namespaces to exclude

3.6.6 JS_URLS_UNNAMED

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

- Default value: `False`

3.6.7 JS_I18N_APPS

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

- Default value: `None`
- Expected: a restricted application list to include in the javascript translation catalog

3.6.8 JS_I18N_APPS_EXCLUDE

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

- Default value: `None`
- Expected: an application list to exclude from the javascript translation catalog

3.6.9 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')
```

3.7 API

3.7.1 `djangojs` – Main package

Django.js provide better integration of javascript into Django.

djangojs.**JQUERY_DEFAULT_VERSION = '1.9.1'**
Packaged jQuery version

3.7.2 `djangojs.views` – Javascript views helpers

This module provide helper views for javascript.

class djangojs.views.**JsInitView**(***kwargs*)
Bases: 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.JsonView

Render the URLs as a JSON object.

class djangojs.views.**ContextJsonView**(***kwargs*)
Bases: 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

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

3.7.3 `djangojs.runners` – Javascript unittest runners

This module provide Javascript test runners for Django unittest.

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*)
Bases: django.core.files.storage.FileSystemStorage

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

3.7.4 djangojs.utils – Miscellaneous helpers

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

It might be splitted in futur releases.

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

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

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

Serialize the context from requests.

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

classmethod as_json(*request*)
Serialize the context as JSON from a given request.

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).

3.7.5 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.

3.7.6 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*)
Include Django.js javascript library in the page

djangojs.templatetags.js.**django_js_init**(*context*, *i18n=True*, *csrf=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

3.7.7 `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 Changelog

3.8.1 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

3.8.2 0.6.5 (2013-03-13)

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

3.8.3 0.6.4 (2013-03-10)

- Support namespaces without app_name set.

3.8.4 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.

3.8.5 0.6.2 (2013-02-18)

- Compatible with Django 1.5

3.8.6 0.6.1 (2013-02-11)

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

3.8.7 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)

3.8.8 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

3.8.9 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

3.8.10 0.3.2 (2012-11-10)

- Optionnal support for Django Absolute

3.8.11 0.3.1 (2012-11-03)

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

3.8.12 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

3.8.13 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

3.8.14 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

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

d