dja Documentation

Release 0.1

Igor 'idle sign' Starikov

Jun 10, 2017

Contents

Dja is a bit of Django framework - its template engine - ported to PHP.

https://github.com/idlesign/dja

Warning: ALPHA stage project. Interfaces might be changed at any moment. Use with care.

CHAPTER 1

Requirements

1. PHP 5.3
2. PHPUnit (to be able to run unit tests)

CHAPTER 2

Table of Contents

Introduction

What's that

Dja is a bit of Django framework, namely its template engine, ported to PHP.

The primary aim of the project is a full-featured and as close to the original code as possible port of Django's template engine, thus introducing this pragmatic template system to PHP developers.

Yet another goal of dja is to make a land for smooth transitions between frameworks in different programming languages, without a need for template rewrites. And, yes, template designers would be grateful and happy too.

Philosophy

Dja is from Python pragmatic world, it won't bother some PHP rules. It is not as if it is an impolite guest - it tries to be respectful whereever it doesn't go in confront with dja's inner dja. It operates upon the *necessary and sufficient* principle. It is wilful, so you can leave anytime you feel you had enough. And that's just how dja goes.

Beware stranger: dja doesn't give a damn for your fancy patterns urge; dja doesn't like ugly namespace notations; dja spits on setters and getters; dja is worse than the Jabberwocky.

So welcome, or goodby, stranger.

Quick start

So you are here to give dja a try. To cut a long story short, I must say that you are not yet ready for it, unless you are already familiar with Django templates syntax.

Templates syntax

First of all, take a look at Django documentation, as it is a good start to get aquainted with templates essentials:

- Syntax overview - https://docs.djangoproject.com/en/1.4/topics/templates/
- Built-in tags and filters - https://docs.djangoproject.com/en/1.4/ref/templates/builtins/

Basic usage example

Now when you're familiar with Django templates[, but not yet ready to quit PHP %)], take a look at a basic dja usage example:

```
<?php

// We'll certainly use dja, so we require it.
require_once 'dja/dja.php';

// Initialize 'TEMPLATE_DIRS' setting to point to our template directory(ies).
Dja::setSetting('TEMPLATE_DIRS', array('/home/idle/my/templates/are/here/'));

// Use shortcut render() method.
echo Dja::render('pages/another_page.html', array('title'=>'My title for another page
↪'));
```

Note: If TEMPLATE_DEBUG setting equals True *Dja::render()* will render pretty page with usefull debug information.

Under the hood the example above does roughly the following:

```
<?php

require_once 'dja/dja.php';

// First we create a template object, passing to it a template source string.
$template = new Template('My template. It counts {{ what }}: {% for item in items %}{
↪{ item }} and {% endfor %}no more. That\'s all, folks!');

// After that we create a context object, passing to it an array of data to be used␣
↪later in template.
$context = new Context(array('items' => range(1, 10), 'what'=>'cows'));

// The final step is to pass our context object to our template object.
$result = $template->render($context);

// Now $result contains a string ready for output.
echo $result;
```

Generic interfaces

Django is a full-blown web-framework whereas dja is only a template engine, but nevertheless some template tags/filters used in dja do require an access to other application parts available in Django, e.g. URL dispatcher, Cache framework, etc. Those parts of Django being rather big and more or less complicated are not ported as a part of dja. Instead dja allows you to “plug in” that functionality, if it is already available to you somehow (as a part of a web-framework or written by yourself), or use dja's generic interfaces which are trying to mimic Django components behaviour.

Note: Dja generic interfaces might be not as efficient as you expected or/and tailored for your needs.

Generic URL Dispatcher

In order to support *url* template tag dja uses a so called URL Dispatcher.

To access dispatcher object use *Dja::getUrlDispatcher()* method.

Dja's generic URL Dispatcher is called *DjaUrlDispatcher* and it roughly mimics Django URL dispatcher, but instead of URLconfs it operates over a set of rules: URL Patterns (regular expressions) to URL alias mappings.

You can pass an array of rules to *DjaUrlDispatcher* using its *setRules()* method.

```
<?php

$dispatcher = Dja::getUrlDispatcher();
$dispatcher->setRules(
    array(
        '~^/comments/(?P<article_slug>[^/]+)/(?P<user_id>\d+)/$~' => 'article_user_
↪comments',
        '~^/client/(\d+)/$~' => 'client_details',
    ));
```

From the example above you can see that array keys are simply regular expressions (with named groups in the first case), and array values are URL aliases. You can address those URLs from your templates using *url* tag: *{% url article_user_comments article_slug user_id %}*.

Supposing that *article_slug* template variable contains a slug associated with a certain article (e.g. my-first-article) and *user_id* contains user identifier (e.g. 33) URL dispatcher should reverse it into */comments/my-first-article/33/*. For more *url* tag examples please refer to Django documentation: https://docs.djangoproject.com/en/dev/ref/templates/builtins/#url

Custom URL Dispatcher

Dja allows you to replace generic URL Dispatcher with a custom one.

To do this one should construct a dispatching class which implements IDjaUrlDispatcher interface, and pass an object of that class into *Dja::setUrlDispatcher()* before template rendering.

Generic Internationalization mechanism

To support internzationalization dja uses simple *DjaI18n* class. It allows, for example, *trans* template tag to localize messages.

To access class object use *Dja::getI18n()* method.

This built-in system uses primitive array storage approach to keep the translations.

One can pass an array of messages indexed by language codes to *DjaI18n* using its *setMessages()* method.

```
<?php

$i18n = Dja::getI18n();
$i18n->setMessages(array(
    'de'=>array(
        'Page not found'=>'Seite nicht gefunden',
    ),
    'ru'=>array(
        'Page not found'=>'  ',
    ),
));
```

To let *DjaI18n* know what languages are supported by your application, you should pass an array with languages definitions to *setLanguages* method:

```
<?php

$i18n = Dja::getI18n();
$i18n->setLanguages(array(
    'de'=>'German',
    'ru'=>'Russian'
));
```

Read more about in-template internationalization at https://docs.djangoproject.com/en/dev/topics/i18n/translation/#internationalization-in-template-code

Custom Internationalization mechanism

Dja allows you to replace generic Internationalization mechanism with a custom one.

To do this one should construct a class implementing IDjaI18n interface, and pass an object of that class into *Dja::setI18n()* before template rendering.

Generic Cache Managers

Dja offers not only several built-in cache managers, but also means to implement that of your own. Caching can be applied both to template parts (*cache* tag) and entire compiled template objects.

To access current cache manager object use *Dja::getCacheManager()* method.

Built-in managers:

- *DjaGlobalsCache*

  The default cache manager, using GLOBALS array to keep cache. Although used as the default, **this mechanism is almost entirely ineffective**, as cached data is not shared among running threads.

- *DjaFilebasedCache*

  This uses filesystem to store cache. Each cache entry is stored as a separate file with serialized data.

- *DjaDummyCache*

  Dummy cache, which doesn't actually cache, but rather implements the cache interface without doing anything. Can be used in develepment.

Cache manager tuning example:

```
<?php

// Let's use filebased cache instead of the default.
Dja::setCacheManager(new DjaFilebasedCache('/tmp/custom/temporaty/path/for/dja/'));
```

Fetch interesting, though not strongly related, information on subject from https://docs.djangoproject.com/en/dev/topics/cache/

Custom Cache Manager

To create your own cache manager please implement IDjaCacheManager interface, and pass an object of that class into *Dja::setCacheManager()* before template rendering.

Dja extras

dja_extras.php file from dja package contains functions and classes absent in Django but proved usefull in Dja.

NaiveIfTemplateNode

This node class can be used as a template for custom naive if-like tags.

This allows *{% if_name_is_mike name %}Hi, Mike!{% else %}Where is Mike?{% endif_name_is_mike %}* and alike constructions in templates.

The class exposes registerAsTag method to quick tag registration within a tag library:

```php
<?php

require_once 'dja/dja_extras.php';

// Let's suppose we're in the template tags library file.
$lib = new Library();

// This registers `if_name_is_mike` if-like tag.
NaiveIfTemplateNode::registerAsTag($lib, 'if_name_is_mike',
    function ($args, $context) {  // This closure will test value on rendering.
        $name = $args[0];  // Our tag accepts only one argument - `name`.
        return $name->resolve($context)=='Mike';
    }
);

return $lib;
```

Extending dja

Sometimes you'll get feeling that dja's built-in filters and tags are just not enough for your needs.

You can create your own tags and filters libraries and use them in your templates with the help of *load* tag.

Libraries

Tags and filters libraries are just php files defining and exporting a *Library* object.

```
<?php

// We create dja library object close to the beginning of the file.
$lib = new Library();

...

// And "export" that object right at the end of file.
return $lib;
```

You can save library file wherever you want, e.g. */home/idle/somewhere/here/mylibfile.php*.

Use DjaBase::addLibraryFrom() to load library into dja.

```
<?php

DjaBase::addLibraryFrom('/home/idle/somewhere/here/mylibfile.php', 'mytaglib');
```

To have access to tags and filters from *mytaglib* library use *load* tag:

```
{% load mytaglib %}

<h1>That's me, {{ name|bongo }}</h1>
```

Custom filters

Adding filters is a rather simple task: just use Library::filter() method to register your filter function (PHP's anonymous function).

```
<?php

// We register filter with name `bongo`.
$lib->filter('bingo', function($value, $arg) {
    // Here `$value` is a value from a template to filter.
    // One may define second `$arg` argument to get filter argument from the template.

    // This filter just adds `-bongo` ending to any filtered value,
    // and doesn't handle any filter arguments.
    return $value . '-bongo';
});
```

Custom tags

Adding tags is a more complex task than adding filters as it involves Node object creation. To add a tag one needs to register it within a library with Library::tag() method.

```
<?php


$lib->tag('mytag', function($parser, $token) {
    /**
     * @var Parser $parser
     * @var Token $token
     */
    $bits = $token->splitContents();  // Here we parse arguments from our tag token.
    // Note that the first argument is a tag name itself.

    if (count($bits)!=2) {
        // Throw an error on argument count mismatch.
        throw new TemplateSyntaxError('mytag takes one argument');
    }

    // Pass tag argument into node object.
    return new MyTagNode($bits[1]);
});
```

Node object metioned above is an object which will take part in tag node rendering. We define our node class somewhere near as follows:

```
<?php

class MyTagNode extends Node {

    private $_arg = null;

    public function __construct($arg) {
        $this->_arg = $arg;
    }

    /**
     * This method will be called each time tag is rendered.
     *
     * @param Context $context Template context.
     * @return string
     */
    public function render($context) {
        return print_r($this->_arg);
    }
}
```

Yii Framework Integration Guide

Dja comes with *YiiDjaController.php* which can be used to allow dja template rendering from Yii (http://www.yiiframework.com/).

To get things done please inherit your application base controller (usually *components/Controller.php*) from YiiDjaController:

```
<?php

// Import dja controller.
Yii::import('application.extensions.dja.dja.YiiDjaController');

class Controller extends YiiDjaController {  // <-- Inherit from YiiDjaController.
    // Your code here.
}
```

Now let your controller action methods call $this->render() as usual, just bear in mind that *view name* param is expected to be in form of a filepath under your [theme] views directory. I.e. to render *{views_dir}/subdir/file.html* dja expects 'subdir/file' from you to be passed as a view name.

Note: Dja will function in debug mode and notify you on template errors if YII_DEBUG = True.

CHAPTER 3

Cheers!

To all the guys from Django team. Without your code there won’t be any dja. You are great! Thank you!

CHAPTER 4

Get involved into dja

Submit issues. If you spotted something weird in application behavior and want to report it you can always do that at https://github.com/idlesign/dja/issues.

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/dja, write your code, whether it should be a bugfix or optimization implementation, and make a pull request right from the forked project page.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish it.

CHAPTER 5

The tip

If dja is not what you want, you might be interested in considering other choices, e.g.:

- h2o-php - http://www.h2o-template.org/
- twig - http://twig.sensiolabs.org/