django-html5-appcache Documentation

Release 0.3.1

Iacopo Spalletti

November 16, 2013

Contents

1 Install **3**

1.1 Installation 3

1.2 Advanced configuration 4

1.3 Changelog 6

2 Usage **7**

2.1 Basic Concepts 7

2.2 Enabling caching in your application 8

2.3 django CMS 9

2.4 Excluding urls from the cache 10

2.5 Web interface 11

2.6 Command line usage 12

3 Indices and tables **13**

3.1 Autodoc 13

Python Module Index **17**

This document refers to version 0.3.1

Application to manage HTML5 Appcache Manifest files for dynamic Django web applications.

While handy and quite simple in its structure, manifest files is quite burdensome to keep up-to-date on dynamic websites.

`django-html5-appcache` try to make this effortless, exploiting the batteries included in Django to discover pages and assets as they are updated by the users.

See *Basic Concepts* for further details.

CHAPTER 1

Install

1.1 Installation

1.1.1 Requirements

- `django>=1.4`
- `lxml`
- `html5lib`

1.1.2 Installation

To get started using `django-html5-appcache` install it with `pip`:

```
$ pip install django-html5-appcache
```

If you want to use the development version install from github:

```
$ pip install git+https://github.com/nephila/django-html5-appcache.git#egg=django-html5-appcache
```

Requirements will be automatically installed.

Run migrate command to sync your database:

```
$ python manage.py migrate html5_appcache
```

Warning: Migrations have been added in 0.3.0. Don't skip this if you are upgrading from 0.2.

1.1.3 Basic configuration

- Add `html5_appcache` to `INSTALLED_APPS`.
- Include in your `URLCONF`:

```
urlpatterns += patterns('',
    url('^', include('html5_appcache.urls')),
)
```

Warning: on Django 1.4+ (or django CMS 2.4+) you may need to use `i18npatterns` instead of `patterns` above, depending on you project layout.

- Enable appcache discovery by adding the lines below in `urls.py`:

  ```
  import html5_appcache
  html5_appcache.autodiscover()
  ```

- Add the middleware just below `django.middleware.cache.UpdateCacheMiddleware`, if used, or at the topmost position:

  ```
  'html5_appcache.middleware.appcache_middleware.AppCacheAssetsFromResponse'
  ```

- Insert `appcache_link` template tag in your templates:

  ```
  {% load appcache_tags  %}
  <html {% appcache_link %} >
   <head>
   ...
   </head>
   <body>
   ...
   </body>
  </html>
  ```

- Enable the cache for your project. Refer to Django `CACHES` configuration.

1.1.4 django CMS integration

See *django CMS installation*.

1.2 Advanced configuration

While no specific configuration is needed to run `html5-appcache`, you can customize its behavior for your own needs with the following parameters:

1.2.1 HTML5_APPCACHE_DISABLE

If you want to keep `django-html5-appcache` installed but you want to disable it temporarely (for debug purposes, for example), set this parameter to `True`: it makes the manifest view return a non-caching manifest file and disables `appcache_link` templatetag. New in version 0.3.0. *Defaults*: `False`

1.2.2 HTML5_APPCACHE_ADD_WILDCARD

If `True` a wildcard entry is added in network section to allow browser to download files not in the `CACHE` section. New in version 0.3.0. *Defaults*: `True`

1.2.3 HTML5_APPCACHE_CACHE_KEY

Name of the cache key.

Defaults: `html5_appcache`

1.2.4 HTML5_APPCACHE_CACHE_DURATION

Duration of the cache values.

Default: `86400` seconds

1.2.5 HTML5_APPCACHE_USE_SITEMAP

`django-html5-appcache` can leverage the `sitemap` application of django to discover the cacheable urls. If you want to disable, you must provide a urls list.

Default: `True`

1.2.6 HTML5_APPCACHE_CACHED_URL

It's possible to provide a list of urls to include in the manifest file as cached urls, if it's not discoverable by the django application (e.g.: it's not managed by django or not linked to any page).

Default: `[]`

1.2.7 HTML5_APPCACHE_NETWORK_URL

You can exclude specific url from being cached by using this parameter. Urls will be excluded by cached urls and set in the **NETWORK** section of the manifest.

Default: `[]`

1.2.8 HTML5_APPCACHE_FALLBACK_URL

It's possible to provide a dictionary of urls to be included in the **FALLBACK** section. Key is the *original* url, value is the *fallback* url.

Default: `{}`

1.2.9 HTML5_APPCACHE_OVERRIDE_URLCONF

When using **django CMS** apphooks, you must provide an alternative urlconf for `django-html5-appcache` to be able to traverse the application urls, due to way apphooks works. See the **django CMS integration** section to know more (WiP)

Default: `False`

1.2.10 HTML5_APPCACHE_OVERRIDDEN_URLCONF

This is used internally by `django-html5-appcache` and should remain to its default value.

Default: `False`

1.3 Changelog

1.3.1 0.3.1 (2013-06-08)

- Fix view-generated manifest

1.3.2 0.3.0 (2013-06-06)

Warning: 0.3.0 introduces migrations. Run `migrate html5_appcache` on upgrade

- Special permissions for management views
- Templatetag to show the chache status and update the manifest (see *Web interface*)
- `HTML5_APPCACHE_DISABLE` parameter to disable manifest file (see *Advanced configuration*)

1.3.3 0.2.2 (2013-06-02)

- Fixes issue with Google Chrome

1.3.4 0.2.0 (2013-06-02)

- Initial release

CHAPTER 2

Usage

2.1 Basic Concepts

`django-html5-appcache` leverages django `cache`, `test` and `signals` frameworks to explore project pages and assets and generate an appcache manifest file.

2.1.1 Manifest file generation

The manifest file is generated collecting all the cached urls and exploring them using the test client to gather asset urls and including them in the manifest itself.

This can be quite resource intensive, so the manifest file is saved in the cache; the view that actually delivers the file manifest to the browser can thus use the cache to serve it with no performance impact.

The manifest file is generated out-of-band using a *django command* so you can execute the command manually or in a cron job. Since 0.3.0 a view is provided, see *Web interface*

Cache invalidation

Whenever a registered model is saved or deleted (see *Enabling caching in your application* on how to enable this for your application), manifest cache is marked as **dirty**; this has no immediate effect on the manifest file served, as the oudated copy is still served.

URL discovery

Using sitemap

`django-html5-appcache` uses the sitemap as a primary mean to discover urls in the web application.

This is a two steps process:

1. get the sitemap and extract the urls declared
2. scrape each url and extract the asset urls

In the scraping phase, the actual HTML of each page is generated and analyed.

Currently `django-html5-appcache` extracts data from `img`, `script` and `link` tags. See `AppCacheAssetsFromResponse` for more in depth details.

See *Markup* on how to customize the assets extraction in your markup.

Customizing urls

Additional to the sitemap method above, you can define your own *custom url list*; in this case, it's your duty to define the list of assets in those urls.

2.2 Enabling caching in your application

`django-html5-appcache` will automatically include your application urls in the manifest file the if you have a sitemap-enabled application; however, to enable cache invalidation, is strongly advised to explicitly enable appcache support in your application.

2.2.1 Basic support

For basic appcache support, you must create a `appcache.py` in your application directory (along `models.py` file), write an `AppCache` class and register it:

```
from html5_appcache import appcache_registry
from html5_appcache.appcache_base import BaseAppCache

from .models import MyModel, AnotherModel

class MyModelAppCache(BaseAppCache):
    models = (MyModel, AnotherModel)

    def signal_connector(self, instance, **kwargs):
        self.manager.reset_manifest()
appcache_registry.register(MyModelAppCache())
```

This code declare support for `MyModel` and `AnotherModel` and hooks `MyModelAppCache.signal_connector` with `post_save` and `post_delete` signals.

Anytime you save or delete an instance of `MyModel` and `AnotherModel` cache will be marked as **dirty**.

2.2.2 Custom urls support

If you don't have a sitemap or you just want to customize the urls in the manifest file, you can add methods to the basic `AppCache` class above:

```
class MyModelAppCache(BaseAppCache):
    ...

    def _get_urls(self, request):
        ...
        return urls

    def _get_assets(self, request):
        ...
        return urls

    def _get_network(self, request):
        ...
        return urls

    def _get_fallback(self, request):
        ...
        return urls
```

- `_get_urls(self, request)`: returns a list of urls to be included in the `CACHE` section of the manifes file;
- `_get_assets(self, request)`: returns a list of asset urls to be included in the `CACHE` section of the manifes file; if you add urls in `_get_urls` method, you have to return the assets in the above urls in this method;
- `_get_network(self, request)`: returns a list of urls to be included in the `NETWORK` section of the manifes file;
- `_get_fallback(self, request)`: returns a dictionary of urls to be included in the `FALLBACK` section of the manifes file; the dictionary key is used as the leftmost url in each manifest row, the value as the rightmost (i.e: the manifest instruct browser to substitute `key` url with `value` url when offline).

`request` object is passed for convenience

2.2.3 django CMS plugins

See *django CMS plugins*.

2.3 django CMS

`django-html5-appcache` supports django CMS out-of-the-box.

django CMS integration delivers support for all the the default plugins; to enable your own plugins see *django CMS plugins* below.

2.3.1 Installation

Plugins

To enable, add the following to `INSTALLED_APPS`:

- `html5_appcache.packages.cms`
- `html5_appcache.packages.filer` (if you use `django-filer`)
- `html5_appcache.packages.cmsplugin_filer` (if you use `cmsplugin_filer`)

Apphooks

If you use applications hooked to django CMS **AppHooks**, you have to write the *AppCache* class; if you use the sitemap method to discover the urls, you **must** add conditional urls loading to the main‘‘urls.py‘‘.

As the scraping uses the internal testserver to deliver the contents, **Apphooks** are not *hooked* so you have to provide an alternate method to attach the urls.

For this purpose use the following snippet:

```
if getattr(settings, 'HTML5_APPCACHE_OVERRIDDEN_URLCONF', False):
    urlpatterns += patterns('',
        url(r'^my-url', include("my-app.urls")),
        ...
    )
```

Where *my-url* is the url where the apphook is attached to, and *my-app.urls* is the urlconf of you application. Repeat for every attached apphook and for every slug they are attached to.

2.3.2 django CMS plugins

To enable cache invalidation for your own plugins, you must create an `AppCache` class for your plugin models too.

The example below is the implementation of an appcache for django CMS text plugin:

```
from html5_appcache import appcache_registry
from html5_appcache.appcache_base import BaseAppCache
from cms.plugins.text.models import Text

class CmsTextAppCache(BaseAppCache):
    models = (Text, )

    def signal_connector(self, instance, **kwargs):
        self.manager.reset_manifest()
appcache_registry.register(CmsTextAppCache())
```

2.4 Excluding urls from the cache

Sometimes you don't want urls to be cached for various reasons (they can pull content from external sites with no way to invalidate the local cache, or they are just non meant to be available offline).

`django-html5-appcache` provides different ways to exclude urls from cache to meet as many usecases as possible.

2.4.1 Configuration

To statically exclude urls from cache or add to the fallback section, use `HTML5_APPCACHE_NETWORK_URL` and `HTML5_APPCACHE_NETWORK_URL`.

2.4.2 AppCache class

In the `AppCache` classes, is it possible to override `BaseAppCache._get_fallback` and `BaseAppCache._get_network` to fine-tune the urls in each section of the manifest file.

2.4.3 Markup

When using `sitemap`, by default every relative URL is considered to be cached, while external URLs are not cached. It's possible to control the behavior of each url by using custom attributes in your tags.

For each `img`, `script` and `link` tag, you can add data-attributes to control how each referenced url is considered:

- *data-appcache='noappcache'*: the referenced url is added to the NETWORK section
- *data-appcache-fallback=URL*: the referenced url is added in the FALLBACK section, with *URL* as a target

2.5 Web interface

Since 0.3.0 `django-html5-appcache` has a small web interface to check the cache status and update the manifest file.

The `appcache_icon` templatag show the cache status icon and hooks it to an ajax call that trigger the manifest update.

2.5.1 Badges

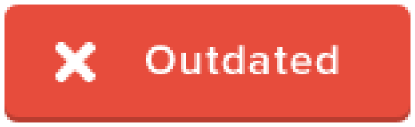

Figure 2.1: Outdated cache status badge

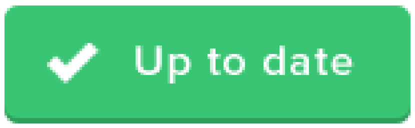

Figure 2.2: Up-to-date cache status badge

2.5.2 Installation

Add the following lines to any template you want the cache status badge to appear:

```
{% load appcache_tags  %}
...
...
{% appcache_link %}
```

2.5.3 Permissions

Both the view that shows the cache status and the view to update the manifest are subject to specific permissions:

- `can_view_cache_status`: required to access the view that show the cache status
- `can_update_manifest`: required to trigger the manifest update

You need to explicitly add these permissions to any user who manages the appcache.

Both the views and the templatetag checks this permissions, so you can actually write your own code to call the views and your code will still be safe.

2.6 Command line usage

`django-html5-appcache` define two commands to control the manifest cache:

2.6.1 update_manifest

`update_manifest` is the command to update the manifest cache.

Run:

```
$ python manage.py update_manifest
```

and your manifest file will be updated.

2.6.2 clear_manifest

Mostly a debugging tool, `clear_manifest` wipe the manifest cache completely.

CHAPTER 3

Indices and tables

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

3.1 Autodoc

class `html5_appcache.appcache_base.`**`AppCacheManager`**

Main class.

`_fetch_url`(*client*, *url*)

Scrape a single URL and fetches assets

`_get_sitemap`()

Pretty ugly method that fetches the current sitemap and parses it to retrieve the list of available urls

`_setup_signals`()

Loads the signals for all the models declared in the appcache instances

`add_appcache`(*appcache*)

Adds the externally retrieved urls to the internal set.

appcache is a dictionary with *cached*, *network*, *fallback* keys

`extract_urls`()

Run through the cached urls and fetches assets by scraping the pages

`get_cached_urls`()

Create the cached urls set.

Merges the assets, the urls, removes the network urls and the external urls

See `BaseAppCache.get_urls()`, `get_network_urls()`

`get_fallback_urls`()

Creates the fallback urls set.

`get_manifest`(*update=False*)

Either get the manifest file out of the cache or render it and save in the cache.

get_network_urls()
: Create the network urls set.

  `*` (wildcard entry) is added when `ADD_WILDCARD` is True (default)

get_urls()
: Retrieves the urls from the sitemap and `BaseAppCache.get_urls()` of the appcache instances

get_version_timestamp()
: Create the timestamp according to the current time.

  It tries to make it unique even for very short timeframes

reset_manifest()
: Clear the cache (if clean)

setup(*request*, *template*)
: Setup is required wen updating the manifest file

setup_registry()
: Setup the manager bootstrapping the appcache instances

class `html5_appcache.appcache_base.`**BaseAppCache**
: Base class for Appcache classes

_add_language(*request*, *urls*)
: For django CMS 2.3 we need to manually add language code to the urls returned by the appcache classes

  Returns list of urls

_get_assets(*request*)
: override this method to customize asset (images, files, javascripts, stylesheets) urls.

  Returns list of urls

_get_fallback(*request*)
: override this method to define fallback urls.

  Returns dictionary mapping original urls to fallback

_get_network(*request*)
: override this method to define network (non-cached) urls.

  Returns list of urls

_get_urls(*request*)
: override this method to define cached urls.

  If you use a sitemap-enabled application, it's not normally necessary.

  Returns list of urls

get_assets(*request*)
: Public method that return assets urls. Do not override, use `_get_assets()`

  Returns list of urls

get_fallback(*request*)
: Public method that return fallback urls. Do not override, use `_get_fallback()`

  Returns dictionary mapping original urls to fallback

get_network(*request*)
: Public method that return network (non-cached) urls. Do not override, use `_get_network()`

  Returns list of urls

`get_urls`(*request*)
Public method that return cached urls. Do not override, use `_get_urls()`

Returns list of urls

`signal_connector`(*instance*, ***kwargs*)
You **must** override this method in you `AppCache` class.

class `html5_appcache.middleware.appcache_middleware.`**`AppCacheAssetsFromResponse`**
Extracts appcache assets from the rendered template.

Currently supports the following tags:

- img: extracts the data in the `src` attribute
- script: extracts the data in the `src` attribute
- link: extracts the data in the `href` attribute if `rel==stylesheet`

It supports custom data-attribute to exclude assets from caching:

- *data-appcache='noappcache'*: the referenced url is added to the NETWORK section
- *data-appcache-fallback=URL*: the referenced url is added in the FALLBACK section, with *URL* as a target

`handle_img`(*tag*, *attrib*)
Extract assets from the img tag

`handle_link`(*tag*, *attrib*)
Extract assets from the link tag (only for stylesheets)

`handle_script`(*tag*, *attrib*)
Extract assets from the script tag

`process_response`(*request*, *response*)
This method is called only if `appcache_analyze` parameter is attached to the querystring, to avoid overhead during normal navigation

`walk_tree`(*tree*)
Walk the DOM tree

Python Module Index

h

html5_appcache.appcache_base, 13

html5_appcache.middleware.appcache_middleware, 15