GUSTAVI

Zeste de savoir

Mise en cache intelligente avec Django

25 avril 2020

Table des matières

Vous êtes en train de construire votre site internet avec Python et Django, tout va pour le mieux du monde ; mais plus votre site grossit et plus des fonctionnalités gourmandes apparaissent, effectuant de plus en plus de requêtes SQL même si vous les optimisez afin d'éviter la casse ?

Que faire quand on a optimisé la génération de ses pages mais que le rendu est encore lent ? Utiliser la mise en cache !

Ce tutoriel est écrit pour Django 1.11 LTS, 2.0, 2.1, 2.2 LTS et 3.0 mais les différences avec les autres versions restent minimes.

1. Présentation

La mise en cache est une technique courante qui consiste à **stocker pendant un temps défini un résultat qui provient d'une opération lourde**. Lors de la demande d'accès à ce résultat, on retourne la valeur que l'on a gardée en mémoire au lieu de relancer l'opération lourde, il en résulte un gain de réactivité (très) important.

Voici un exemple du fonctionnement du cache sur un site web :

1. le site est désormais hors ligne.

FIGURE 1.1. – Fonctionnement du cache sur une page web

2. Les différentes implémentation du cache

Django nous propose nativement un système de cache assez poussé. Celui-ci vous permet d'utiliser différentes plate-formes (aussi appelées **backend**) afin de se souvenir des valeurs que vous voulez réutiliser. On compte notamment :

- Solutions très rapides : via la RAM
  - memcached , la référence en la matière. L'outil va stocker directement dans la mémoire vive les valeurs afin de garantir un temps d'accès optimal. De plus, l'architecture distribuée[2] de memcached permet d'utiliser des machines distantes afin de répartir la mise en cache, tout est géré automatiquement.
  - Si vous n'êtes pas en mesure d'installer, de configurer et d'utiliser memcached (par exemple si vous n'avez pas les droits root sur le serveur) mais que vous voulez tout de même avoir un stockage en RAM pour un accès plus rapide, Django possède un *backend* prévu à cet effet .
- Solutions un peu moins rapides : via le disque
  - Dans une base de données ;
  - Dans des fichiers temporaires .
- Autres solutions :
  - Un cache qui ne cache rien à utiliser sur des versions de développement par exemple ;
  - Un cache personnalisé si vous avez des besoins très spécifiques.

C'est à vous de faire le choix concernant le *backend* que vous voulez utiliser en fonction de vos besoins, mais sachez qu'il est possible d'utiliser plusieurs *backends* sur un même site ! Par exemple, les éléments peu nombreux et utilisés très souvent pourront être stockés dans la mémoire vive alors que les éléments plus volumineux, nombreux et moins souvent accédés pourront se trouver sur le disque (sauf si vous avez à votre disposition 256Go de RAM).

3. Différents niveaux de mise en cache

De plus, Django nous permet différent seuils de granularité :

- **Une mise en cache globale du site :** chaque page sera cachée pendant un certain temps.
- **Une mise en cache par vue :** seulement certaines vues seront cachées par l'utilisation d'un décorateur Python.

2. Vous pourrez trouver une description de l'architecture distribuée ici .

— **Une mise en cache au niveau des *templates* :** c'est la méthode la plus fine, qui permet de cacher uniquement certains bouts dans certains modèles HTML.

C'est en se fondant sur cette dernière technique que nous allons réaliser une mise en cache intelligente.

4. Exemple : système de notifications

5. Situation initiale

Prenons un cas concret : vous êtes en train de réaliser une application pour votre site qui s'occupe de gérer l'envoi de messages entre les membres de votre site. Vous avez intégré la notification de nouveaux messages via une icône présente sur tout le reste de votre site.

Cependant, **la requête pour récupérer le nombre de nouveaux messages est lourde et prend 3 secondes**. Sachant que cette requête est effectuée sur chaque page puisque vous vous servez le l'icône pour afficher le nombre de nouveaux messages, **toutes les pages de votre site mettent 3 secondes de plus à charger à cause de cette requête**.

Mettre du cache pour combler certaines lenteurs peut être une mauvaise idée : si votre site est lent c'est peut-être lié un souci de conception. En effet, un site doit pouvoir fonctionner normalement sans cache, ce dernier étant là pour des questions d'**optimisations**. Avant la mise en place d'un cache il est nécessaire vérifier qu'il est impossible d'améliorer les performances sans cache et que ce cache va vraiment apporter un gain, ce qui ne serait pas le cas sur une page lente que personne ne va voir par exemple.

Heureusement, comme on l'a vu précédemment, Django nous propose un système de mise en cache au niveau des *templates* HTML. Et si nous utilisions cet outil afin d'accélérer la génération de nos pages ?

6. Mise en place du cache

Rendons nous dans notre *template* où le code gourmand est appelé, et mettons ce résultat en cache :

Après avoir configuré correctement le système de cache (n'utilisez pas le cache qui ne cache pas sinon ça risque de ne pas fonctionner), on remarque que la première génération de cette page prend toujours 3 secondes mais que **les suivantes sont rendues quasiment instantanément** !

Cependant il y a un problème. En effet, si vous vous connectez sous le nom d'un autre utilisateur, alors vous aller voir le bloc qui a été caché pour le premier utilisateur à avoir entraîné la mise en cache. Ainsi il se peut très bien que vous ayez une notification de nouveau message alors que c'est en fait le nombre de messages pour le premier utilisateur qui avait été mis en cache !

7. Indépendance du cache

Django nous permet de palier à ce problème : il nous suffit de trouver un élément unique qui diffère et de le passer à la balise de mise en cache pour que celle-ci stocke les différentes valeurs en fonction de cet élément unique. Dans le cadre de nos utilisateurs, on peut dire que le pseudonyme est un bon élément unique :

```
```

Ainsi, chaque utilisateur aura sa propre mise en cache de son nombre de nouveaux messages. Ouf !

> **Astuce** : vous pouvez stocker un bloc unique pour chaque utilisateur et un autre dans le cas d'un utilisateur non connecté avec `{% cache 1800 menu user.pk|default_if_none:"0" %}`. Ici on met en cache pendant 1800 secondes un bloc qui s'appelle « menu » et dont la clé unique est le `pk` de l'utilisateur. Si l'utilisateur n'est pas connecté alors ce `pk` vaudra `None` et et on le remplace par 0 (les `pk` commencent à 1) avec le filtre default_if_none . Ainsi tous les visiteurs non connectés verront le bloc avec `"0"` alors que les utilisateurs authentifiés auront leur propre bloc. Cela peut être utile sur un menu où tous les utilisateurs n'ont pas accès aux mêmes parties du site.

8. Mise à jour intelligente du cache

9. Persistance indésirable

Il nous reste cependant un dernier petit problème à régler :

Considérons que je viens de recevoir un nouveau message ; la valeur mise en cache m'indique bien que j'ai un nouveau message puisque c'est la première page que je génère, tout fonctionne comme prévu.

Je lis mon message et éventuellement répond à son auteur avant de continuer paisiblement ma navigation sur le site. Cependant, je remarque que le nombre de nouveaux messages reste bloqué à 1 ! En effet, la valeur a été mise en cache pendant un certain temps (120 secondes dans notre exemple précédent) et ne sera pas actualisée avant que ces 120 secondes se soient écoulées !

Cet effet est gênant pour tous les utilisateurs de votre site, mais comment faire pour éviter ce cas de figure tout en gardant les bénéfices de la mise en cache ?

FIGURE 9.2. – Un exemple de cache non actualisé : le compteur (en cache) affiche 11 notifications alors qu'il n'y a qu'une seule

10. Péremption automatique

Afin de solliciter un appel frais à la fonction de génération de notifications, il nous faudrait invalider le contenu qui a été précédemment mis en cache. Pour cela, nous allons devoir utiliser l'accès « bas niveau » au cache de Django.

Le cache fonctionne comme un dictionnaire avec un système de clés/valeurs ; pour supprimer un élément il nous faut donc retrouver sa clé. Les clés utilisées au niveau du cache sont assez bizarres, considérons le bloc suivant :

```

```

La clé associée sera la suivante :

```

```

On y retrouve le nom du bloc ainsi qu'un *hash* MD5 créé à partir des arguments passés au bloc. Pour retrouver facilement cette clé, on peut utiliser la fonction `make_template_fragment_key` :

```

```

Il nous reste maintenant à supprimer cette clé du cache lors des actions qui pourraient potentiellement mettre à jour ce cache. Par exemple, si l'on reprend notre cas, on aimerai bien que le cache soit mis à jour lors de la consultation du nouveau message :

```

```

Il vous reste à mettre en place cette demande explicite de suppression de contenu mis en cache à chaque endroit où ce contenu aurait pu en effet être modifié. Ainsi, les utilisateurs sont toujours prévenus en temps et en heure, en plus si aucun changement n'a été détecté la valeur peut directement être lue dans le cache !

Vous êtes désormais en mesure d'utiliser la mise en cache de Django sur votre site web sans y retrouver les effets néfastes : la mise à jour différée des données.

Sources :

— La documentation officielle de Django 3.0 ;
— Mon expérience personnelle (MicroJoe) lors de la mise en place d'un système de cache pour Progdupeupl .

> *i* Je rappelle que ce contenu provient initialement de Progdupeupl et a été écrit par MicroJoe sous la licence CC BY . Je le remercie de m'avoir laisser le reprendre et le mettre à jour .

Merci à artragis et spacefox pour les remarques toujours pertinentes qu'ils ont eues.