Meetic
Création du document : 12/02/2009
Auteur : Mathieu Perrin

Documentation

Module de liste

Nom de code : mlist

Révisions :

Auteur	**Date**	**Notes**
Mathieu Perrin	12/02/09	Création du document

Sommaire

Table des matières

Introduction

Ce document est une présentation technique du module *mlist* permettant de gérer des listes. Elle est donc destinée aux développeurs devant mettre en place des listes.

Sujets abordés

Le principe général du module sera exposé de façon à ce qu'un développeur comprenne rapidement la philosophie du module. Des cas d'utilisation simples seront expliqués.

Sujets non abordés

Les cas d'utilisation complexes ne sont pas abordés par ce document dans un premier temps. Ceci pour la raison que l'auteur de ce document n'est pas l'auteur de la librairie et que certains points restent encore obscurs.

Fichiers concernés

Les fichiers concernés par cette documentation sont situés sous la hiérarchie de dossiers : libs/sdk/list/mlist.inc

Vocabulaire

Objet : entité composant une liste. Exemple : un membre

Identifiant : attribut unique à un objet. Il existe une bijection entre identifiants et objets. Exemple : un aboid.

Liste : suite ordonnée d'entités. C'est ce qui apparaît sur l'écran de l'utilisateur. Elle a une valeur sémantique pour l'utilisateur. Exemple : liste de membres favoris. Selon le contexte, peut également indiquer une liste d'identifiants ou une liste d'objets.

Liste d'identifiants :suite ordonnée d'identifiants représentant des objets de même type. Exemple : liste d'aboid.

Liste d'objets : suite ordonnée d'objets de même type. Exemple : liste de membre.

Instance de liste : représentation informatique d'une liste.

Principe général du module

Caractéristiques principales

Le module *mlist* permet la gestion de listes tels que les favoris d'un utilisateur ou ses messages non lus. Il n'y a pas de taille maximale aux nombres d'objets que peut contenir une liste.

Le module propose une gestion de cache permettant de limiter le nombre de requêtes et donc la charge d'une base de données par exemple. A noter que c'est la liste d'identifiants qui est mise en cache et non la liste d'objets, de façon à limiter la taille du contenu du cache.

Composants

Plusieurs composants forment une instance de liste. On peut les regrouper en 5 catégories : constructeurs principaux, filtres, vues, enrichisseurs, et navigateurs.

[schéma nécessaire]

Constructeur principal

Il n'y en a qu'un seul par instance de liste. Ce composant définit une fonction qui permet d'obtenir les identifiants des objets de la liste.

Filtre

Il peut y avoir plusieurs filtres par instance de liste. Un filtre permet de supprimer certains identifiants que le constructeur principal aurait ajouté à la liste d'identifiants. Par exemple, dans une liste de membres à qui ont a écrit, on peut vouloir retirer à la fois ceux qui l'on a blacklistés et ceux qui ne correspondent pas à ses préférences de recherche.

Vue

Une instance de liste peut avoir plusieurs vues. Une vue met en relation les différents filtres qui doivent être appliqués sur la liste d'identifiants de sortes à créer la liste d'objets souhaités. En relation avec l'exemple précédent, on peut vouloir créer 4 vues, une qui ne filtre rien, une qui filtre les membres blacklistés, une filtrant ceux qui ne correspondent pas aux préférences de recherche, et une dernière filtrant à la fois ceux blacklistés et n'étant pas dans les préférence de recherche.

Enrichisseur

Un enrichisseur est associé à une vue. On peut définir plusieurs enrichisseurs par instance de liste pour les associer à des vues différentes. Ce composant permet, à partir d'un identifiant, de créer un objet. Par exemple, on peut enrichir un aboid de la photo et du nom du membre correspondant.

Navigateur

C'est ce composant qui lie un enrichisseur à une vue.

Affichage d'une liste

Le module fournit 4 informations : le nombre d'objets dans la vue, les objets eux mêmes,sous forme de liste ordonnée, les objets sous forme de liste projetée, et des informations sur la pagination.

Nombre d'objets

Un entier

Liste ordonnée

La liste d'objets

Liste projetée

On définie la liste projetée comme ayant un nombre d'items égal au nombre d'informations contenues par un objet.

On regroupe dans un item toutes les informations identiques de chaque objets.

Cette liste est spécifique à Yats.

[schéma nécessaire]

Pagination

Des informations sur les pages suivantes et précédentes.

[schéma nécessaire]

Définition des méthodes

Les méthodes principales permettant de créer les composants d'une liste sont présentés, ainsi que les arguments principaux. Pour le détail de ces méthodes, se reporter à la documentation auto-générée du code source.

[chapitre en cours de rédaction]

Le constructeur

Nom de la méthode : mlist_new

Arguments :

$listId : une liste à un nom qui permet de l'identifier en session. Si toutes vos listes ont des noms différents, la session va grossir. Si vos listes ont toutes le même nom, seulement la dernière restera en session.

$func : Le nom de la fonction, sous forme de chaine de caractère qui renvoie la liste d'identifiants.

$args :Les arguments requis par $func, sous forme de tableau

Les filtres

Une fonction qui prend en entrée une liste d'identifiants et qui retourne une liste d'identifiants.

Les vues

On peut définir un ttl à une vue.

Les enrichisseurs

Une fonction qui prend une liste d'identifiants en entrée et qui doit retourner un tableau associatif dont les clés sont les identifiants et les valeurs les objets associées.

Les navigateurs

Définissent l'ordre d'affichage des objets dans la vue : paginé, mélangé, normal. Peu d'information à ce sujet.

Optimisations

L'optimisation consiste en général à des compromis à faire. Le cache est la principale source d'optimisation de ce module.

Le cache

Le constructeur principal de la liste demande d'assigner un ID à une liste. Cela permet de stocker chaque liste dans le cache et de les retrouver, évitant ainsi de nouvelles requêtes. Cependant, de nombreuses listes peuvent faire grossir énormément le cache. En assignant intelligemment des ID similaires à certaines listes il est possible de limiter la taille du cache.

Gestion des temps en cache

Différentes méthodes (mlist_new, mlist_viewAdd) ont en entrée un agument nommé $ttl (time to live) qui spécifie combien de temps, en secondes, le contenu doit rester en cache. Une valeur de 0 indique que le cache n'est jamais invalidé.

Certaines listes peuvent être définies comme « asynchrones » quand elles sont liées à des événements extérieurs dont l'utilisateur n'a pas à être informé immédiatement (les visites des autres membres par exemple). Dans ce cas, on peut définir un ttl positif (par exemple équivalent à 10 minutes pour les visites des autres utilisateurs).

Certaines listes peuvent à contrario être définies comme « synchrones » quand leur contenu peut se mettre à jour à n'importe quel moment et que l'utilisateur en est averti (par exemple une liste de favori doit être mise à jour uniquement lorsque l'utilisateur ajoute ou supprime une personne de la liste). Dans ce cas, un ttl nul permet de garder la liste en cache tant que rien ne vient la modifier. Un appel sous certaines conditions à la méthode mlist_refresh permet de mettre à jour le cache uniquement lorsque cela devient nécessaire.

Conclusion et améliorations possibles

Ce module de liste permet de facilement gérer de multiples listes d'objets. Différents choix

d'optimisation sont possibles ce qui permet l'adaptation à différents cas de figure.

Cependant, il est clair que certains points pourraient être améliorés, tel que la documentation des méthodes, car l'appel à certaines méthodes est un peu obscure, ce qui rend l'utilisation du module peu intuitif. Dans certains cas il apparaît également superflu de créer tous les composants (vue, enrichisseur, filtre) qui ne sont pas toujours utiles.

A contrario des différents composants qui permettent une flexibilité dans la création d'une liste, il apparaît que les informations renvoyées par le module pour l'affichage de la liste ne sont pas flexibles du tout. Des 4 informations renvoyées par le module, aucune n'est accessible séparément des autres, ce qui implique la création et le transfert de données souvent inutiles. De plus, les informations sur la pagination sont figées : il n'est pas possible de définir le type d'affichage que l'on souhaite pour la pagination. L'information spécifique au moteur d'affichage Yats contraint à l'utilisation de ce moteur. Il faudrait déporter cette information pour rendre le module plus générique encore.

Le module étant écrit en PHP 4 sans classe, Un compromis à la réécriture sous forme de classe serait de créer une sur-couche à ce module pour que le module devienne un peu plus intuitif au développeur.