ZiAPI Specifications

beauze_h
courta_c
fauvet_l
boquet_t

March 18, 2008

Contents

Chapter 1

Chaine de Production

On peut decouper les grandes phases de fonctionnement du serveur en trois parties

1. L'initialistation du serveur
2. La chaine de traitement des requetes
3. L'arret du serveur

1.1 La chaine de production

Le traitement des requetes peut se scinder en 7 etapes distinctes, ce qui permettera aux modules de se greffer entre ces differentes etapes :

1. Connection d'un client, avant de recevoir des donnees de ce dernier
2. Reception et parsing d'une requete HTTP
3. Reception d'un enventuel contenu post header (multipart/form-data par exemple)
4. Creation d'un header de reponse
5. Envoie du header de la reponse
6. Envoi du corps de la reponse
7. Fin du traitement de la requete

Le decoupage du traitement d'une requete en differentes etapes nous permet donc de venir rajouter des modules avant ou apres ces etapes, qui effectueront des diverses actions. Ces differents modules pourront agir sur une ou plusieurs etapes de la chaine de production.

1.1.1 Connection d'un client

Lors de la connection d'un client, son socket est associe a une tache, puis cette tache ajoutee a la pile des taches a executer. Cette operation est realisee par le thread principal dans le cas d'une nouvelle connection, ou bien par le Pool Manager dans le cas d'un client en KeepAlive qui enverrait une nouvelle requete. Le traitement de cette requete se fera par la suite, dans un thread dedie a cette tache.

1.1.2 Reception et parsing d'une requete HTTP

A partir de cette operation, nous sommes maintenant dans un thread specialise dans l'execution des taches. La tache a ete retiree de la pool des taches a executer, et son traitement commence par la reception du header de la requete. Les donnees sont donc recu, concatenees dans un buffer, puis envoyees au parser HTTP, qui creera une instance d'un objet HttpRequest, contenant tout les parametres de la requete. A noter que la reception de donnees est non bloquante afin de rendre possible la detection d'un eventuel timeout.

1.1.3 Reception d'un contenu post-header

//FIXME

1.1.4 Creation d'un header de reponse

A partir de l'instance de HttpRequest cree lors du parsing de la requete, une instance de la classe HttpResponse est generee. C'est cette classe qui permettra de renvoyer la reponse au client. Selon si la requete demande un fichier ou un listing de repertoire, l'instance de HttpResponse sera differente. La recuperation du conteu est assure par la methode virtuelle pure getContent, qui sera implementee dans les classes HttpResponseFile, HttpResponseDir, ou bien HttpError. De cette facon, la lecture et l'envoie de la reponse peut se faire de maniere uniforme. Ces sous classes remplissent egalement les headers de reponse en adequation avec leur fonctions. Certains champs sont ajoutes de facon commune a toutes les reponses, comme le champ "Server".

1.1.5 Envoie du header de la reponse

Pour envoyer le header de reponse, la liste des champs de ce header est iteree, puis ajoutee dans un buffer, qui sera tout simplement envoye au client.

1.1.6 Envoie du corps de la reponse

Le corps de la reponse est obtenu grace a un stream, le dit stream etant construit dans la sous classe de HttpResponse adaptee a la requete. Chaque partie de ce stream est ajoutee dans un buffer, puis le buffer est envoye au client, ce jusqu'a la fin du stream.

1.1.7 Fin du traitement

Cette etape est appelee quelquesoit le resultat de la requete. Elle s'occupe de reinitialiser la tache afin qu'elle puisse etre replacee dans la pool, ferme la connection avec le client dans le cas ou ce client ne demande pas de KeepAlive. Dans le cas contraire, le client est ajoute a la liste des sockets en KeepAlive.

1.2 Interfacage des modules

Chapter 2

Intégration des modules dans le serveur

2.1 Chargement des modules

Le chargement d'un module s'effectue suivant le shema suivant:

1. Lecture de l'emplacement du module depuis la configuration.
2. Verification des symboles exportés.
3. Identification des interfaces utilisées.
4. Création et déploiement des instances correspondantes.

2.1.1 Configuration

Afin qu'un module soit dynamiquement chargé il doit être présent dans le fichier *modules.conf* (inclus depuis *zia.conf*). Il s'agit d'un arbre XML contenant les chemins vers les librairies dynamique (*so* sous Linux ou *dll* sous Windows).

2.1.2 Verification des symboles

Un certain nombre de symboles doivent être obligatoirement exportés afin de permettre un socle commun permettant l'intéraction avec le serveur:

create permet de créer une nouvelle instance du module (plus precisement de sa classe principale).

destroy détruie l'instance du module passée en paramètre.

name retourne le nom du module.

Si l'un de ces symboles n'est pas correctement exporté le module ne pourra pas être chargé et une erreur sera retournée.

2.1.3 Identification des interfaces

L'API propose diverses interfaces pouvant être implémentées par les modules, ces interfaces contiennent diverses méthodes qui seront appelées à des points stratégiques du serveur. Elles sont toutes optionnelles et peuvent être héritées par les modules pour supporter tel ou tel évenement.

Ces héritages sont donc utilisés conjointement avec des *dynamic_cast* pour déterminer quels évenements sont supportés par le module et ainsi l'inscrire aux différents points clés.

Chapter 3

Outils Accessibles depuis les modules

bite chatte cul

Chapter 4

Description d'un module

Chapter 5

Types de Modules

On peut categoriser les modules en 5 types differents (un module peut etre de plusieurs types a la fois)

1. Module Server qui pourra agir au lancement et a la fermeture du serveur
2. Module Accept qui agira sur la creation d'un socket
3. Module Content qui agira sur le contenu de la reponse
4. Module Build qui agira apres la creation de l'en tete de reponse
5. Module Receive qui pourra agir avant ou apres la reception de donnees du client

Chapter 6

Example(s)