Architecture du code CHATHACK

Packages

Frame

Le package **Frame** possède une classe static dans l'interface Data pour chaque trame correspondant au RFC du protocole. Cela nous permet de différencier les informations de chaque type de trames utilisées. On utilise une classe Enum **StandardOperation** pour regrouper les Code_OP définis aux actions des trames. L'interface Frame permet à partir d'une classe de Data de créer une objet Frame via une factory ainsi qu'une méthode buffer() qui renvoie un ByteBuffer correspondant à la trame. On retrouve donc une classe qui implémente **Frame** pour chaque trame nécessaire à notre protocole. On utilise un Visitor avec la classe **FrameVisitor**, il en existe une instance différente pour le serveur et chaque client.

Reader

Le package **Reader** est composé essentiellement de l'interface **Reader** qui permet, à partir d'un ByteBuffer, de récupérer un objet Data (au final) qui correspond aux informations de la trame liée au ByteBuffer. Nous utilisons une classe **SelectOpCodeReader** qui permet la sélection du Reader à utiliser via l'Op_Code contenu dans le ByteBuffer. **Reader** est une interface paramétrée, nous avons divisé les classes qui l'implémentent dans deux "sous-packages": - "**basics**" qui regroupe les Readers de types simples qui renvoie des objets du type correspondant - "**data**" qui utilise les Readers basiques pour renvoyer des objets Data.

NonBlocking

Ce package correspond au serveur et au client. Le serveur est codé dans la classe**ServerChatHack**, le ServerSocketChannel est à l'état OP_ACCEPT pour pouvoir accepter les clients qui se connectent. Il possède aussi un SocketChannel en OP_CONNECT pour se connecter au serveur MDP. Nous avons deux factory dans la classe **ServerChatHack** qui va crée des visiteurs différents. Il y en aura un qui sera utilisé pour la communication client serveur et un autre pour la communication serveur, cela évite que le client puisse communiquer s'il n'est pas encore connecté. Lors d'une connexion, le serveur envoie une demande de vérification du login au serveur base de données et attend la réponse en état OP_READ. On garde la réponse dans une map<Long, DataMdp>, la classe DataMdp va nous permettre de garder la demande du client et la réponse du serveur.En cas de validation, on enregistre le client dans une map<String, SelectionKey> avec en clé son login et en valeur sa SelectionKey et on renvoie un ACK pour la connexion au client initial. Le client lié à la classe **ChatChaton** se connecte en précisant son adresse, le port d'écoute et son login. Il est tout d'abord en état OP_Connect puis en OP_READ en attendant la réponse du serveur. En cas de refus, on ferme la socketchannel. Suite à la connexion d'un client on peut envoyer des messages public, c'est à dire qu'on envoie une trame global au serveur qui va la transmettre à tous les clients connectes au serveur. Le client pourra envoyer des demandes de connections privées et ça sera au serveur de relier les demandes et les réponses entre les client (ceci est expliqué plus en détail dans le protocole). Si les clients décident d'établir une connexion ça sera au client de se connecter à un autre et une fois connectés, ces clients pourront s'envoyer des trames sans intervention du serveur.

Diagramme UML Simplifié