SYM Laboratoire 2 - Protocoles applicatifs

Thomas Léchaire, Michaèel Brouchoud et Kevin Pradervand 16.11.2018

Introduction

Dans ce laboratoire, il est demandé de tester les techniques de programmation réparties asynchrones dont

- La transmission asynchrone
- La transmission différée
- La transmission au format JSON
- La transmission au format XML
- La transmission avec GraphQL en JSON
- La transmission compressée

Questions

4.1 Traitement des erreurs

Les interfaces AsyncSendRequest et CommunicationEventListener utilisées au point 3.1 restent très (et certainement trop) simples pour être utilisables dans une vraie application : que se passe-t-il si le serveur n'est pas joignable dans l'immédiat ou s'il retourne un code HTTP d'erreur ? Veuillez proposer une nouvelle version, mieux adaptée, de ces deux interfaces pour vous aider à illustrer votre réponse.

Tout d'abord, cela limite la souplesse du code, en effet, nous nous sommes retrouvé avec beaucoup de code dupliqué, une solution serait celle proposée dans notre code, à savoir de faire en sorte que le AsyncSendRequest délègue l'envoi de la requête à un ARequestOperation. Cela à permis de factoriser le code entre les classes AsyncRequestOperation et CompressedRequestOperation.

Si on prend uniquement le principe de base énoncé au point 3.1, il est facile de constater qu'il manque un élément principal permettant de vérifier l'état de la connectivité du téléphone mobile avant d'effectuer la requête. Le plus simple serait de vérifier celui-ci avant d'effectuer l'envoi pour être ainsi sûre qu'il soit possible de le faire et dans le cas contraire de sauver la requête puis d'attendre le moment opportun pour retenter un envoi après une certaine période d'attente. (avec une méthode isConnectedToNetwork() par exemple)

En ce qui concerne les erreurs HTTP, cela dépends du code erreur. Une erreur 404 aurait tendance à utiliser le même principe que celui énoncé précédement. Pour une erreur 500 ou un coupure réseau, il faudrait ajouter un méchanisme de boucle pour relancer la requête N fois si une erreur est detectée avant d'afficher un message d'erreur ou une demande de vérification de la connexion à l'utilisateur suite aux nombreuses tentative par exemple.

4.2 Authentification

Si une authentification par le serveur est requise, peut-on utiliser un protocole asynchrone ? Quelles seraient les restrictions ? Peut-on utiliser une transmission différée ?

Il y a deux restriction importantes à prendre en compte. Dans les deux cas, il faut vérifier la connexion au server avant de transmettre la requête. Une requête asynchrone ou différée n'a pas vraiment lieu d'être étant donnée que l'activité suivante accès à la zone de login (après authentification) dépend entièrement de la réponse du server. Donner l'accès sans le réseau (différée) ou avant avoir reçu la réponse est complètement contraire au principe d'authentification.

Les restrictions potentielles seraient l'attente de la réponse (dans les 2 cas) et de la vérification de l'authentification avant toutes autres choses.

4.3 Threads concurrents

Lors de l'utilisation de protocoles asynchrones, c'est généralement deux threads différents qui se préoccupent de la préparation, de l'envoi, de la réception et du traitement des données. Quels problèmes cela peut-il poser ?

Il pourrait potentiellement y avoir des problèmes d'accès concurent à certaines ressources. 2 Threads pourraient modifer l'UI et entrer en conflit (exemple: si un thread écrase les modifications de l'autre).

Les asynctasks s'exécutent les unes après les autres contrairement aux Threads qui eux peuvent s'éxecuter en parallèles. Les asynctasks posent donc moins de soucis. Les threads quant à eux sont plus problématiques et il faut gérer les accès aux sections critiques avec des synchronise sur les variables potentiellement problématiques.

4.4 Ecriture différée

Lorsque l'on implémente l'écriture différée, il arrive que l'on ait soudainement plusieurs transmissions en attente qui deviennent possibles simultanément. Comment implémenter proprement cette situation (sans réalisation pratique) ? Voici deux possibilités :

- *Effectuer une connexion par transmission différée*
- *Multiplexer toutes les connexions vers un même serveur en une seule connexion de transport.*

Quels avantages peut-on espérer de ce multiplexage ? Dans ce dernier cas, comment implémenter le protocole applicatif ? Comment doit-on planifier les réponses du serveur lorsque ces dernières s'avèrent nécessaires ?

Comparer les deux techniques (et éventuellement d'autres que vous pourriez imaginer) et discuter des avantages et inconvénients respectifs.

Connexion par transmission différée:

- Pratique lors de volume plus important
- Lors de coupures on ne retransmet que les requêtes non resolues
- Limitation dans la bande passante
- Pratique lors de connexion faibles
- Limitation de l'utilisation du réseau

Multiplexage des connexions:

- Grosseurs des requêtes.
- Beaucoup de données à retransmettre lors de coupures réseau.
- Plus pratique avec des petits volumes de données.
- Structures complexes des requêtes et des réponses.
- La sérialisation et la désérialisation peuvent être complexes lors de multiplexages de requêtes importantes (nombreuses).
- La réponse doit être décomposées et retournées au différentes activités.

- Le traitement de la requête par le server peut être long.
- économie de batterie

Pour implémenter le protocol applicatif, il faut bien identifier les requêtes, avec un id par exemple afin de pouvoir traiter les réponses séparément. Le xml ou le Json sont deux manières de structurer ces requêtes, par exemple avec un balise de début et de fin en xml. en json on pourrait mettre chaque requête dans un sous-tableau.

- Le plus simple serait de faire de l'Active Poll ou le serveur profiterait pour quittancer les données reçues ou d'utiliser le service de notification et notifier l'utilisateur une fois toutes les données reçues.

4.5 Transmission d'objets

a. Quel inconvénient y a-t-il à utiliser une infrastructure de type REST/JSON n'offrant aucun service de validation (DTD, XML-schéma, WSDL) par rapport à une infrastructure comme SOAP offrant ces possibilités ? Est-ce qu'il y a en revanche des avantages que vous pouvez citer ?

L'inconvéniant c'est qu'il n'y a aucun contrôle du format de transmission des données contrairement au SOAP qui impose de définir une structure. C'est alors au développer de faire attention lors de la sérialisation et désérialisation des requêtes / réponses.

Le fait de ne pas avoir de structure à l'avantage d'avoir un poids moins elevée et d'avoir une structure moins complexe. SOAP, XML proposent certes un DTD, mais cela alourdi la requête par un grand nombre de balises qui ne sont pas forcément utiles à la requête sur le server.

b. L'utilisation d'un mécanisme comme Protocol Buffers8 est-elle compatible avec une architecture basée sur HTTP ? Veuillez discuter des éventuelles avantages ou limitations par rapport à un protocole basé sur JSON ou XML ?

Protocol Buffers 8 est compatible avec une architecture basée sur HTTP. C'est une manière plus légère de sérialiser et d'avoir une vérification de la structure des données.

Pour la transmission des requêtes sur le réseau, il faut utiliser l'entête "application/octet-stream", il faut faire attention au fait que et le client et le server doivent pouvoir traiter et gérer la requête/réponse correctement.

Par rapport à XML Protobuff est plus léger, plus rapide, moins ambigu. Protocol Buffer n'est pas conseillé pour modéliser un document basé sur du texte, mais plutôt pour des structures type objets. XMl sera plus compréhensible.

Par rapport à Json il propose la validation en plus.

c. Par rapport à l'API GraphQL mise à disposition pour ce laboratoire. Avez-vous constaté des points qui pourraient être améliorés pour une utilisation mobile ? Veuillez en discuter, vous pouvez élargir votre réflexion à une problématique plus large que la manipulation effectuée.

L'API GraphQL: Sans la documentation, il peut être difficile de connaître tous les champs disponibles. Une Api graphQL avec beaucoup de champs devrait proposer un nombre de méthodes plus grandes afin d'éviter à la personne utilisant d'api d'avoir à écrire les 50 champs qu'elle a besoin de récupérer. De plus on pourrait imaginer les problèmes qu'il peut y avoir dans le cas ou le nom d'un des champs viendrai à changer. L'application pourrait ne plus fonctionner. L'utilisation des méthodes peut-être quelque chose à préconiser.

4.6 Transmission compressée

Quel gain peut-on constater en moyenne sur des fichiers texte (xml et json sont aussi du texte) en utilisant de la compression du point 3.4 ? Vous comparerez vos résultats par rapport au gain théorique d'une compression DEFLATE, vous enverrez aussi plusieurs tailles de contenu pour comparer.

Envoie de la chaine Hello avec une réponse de taille :

	Hello	**Hello World Comment ca va ?**	**abcdefghijklmnop**
Compressé	467	483	475
Non Compressé	777	799	788
Ratio	60%	60%	60%

Donc environ 40% de gains. Vu que l'algortihme de gzip s'appuie sur l'algorithme de deflate. Il a été trouvé que le gain maximal de cette algorithme soit de 80% Quick Benchmark. Il est donc très efficace.