HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Laboratoire 2 : Protocoles applicatifs

Systèmes mobiles

Auteur : Spinelli Isaia et Simonet Yoann
Prof : Dutoit Fabien
Assist. : Christophe Greppin
Date : 11.10.2019
Classe : SYM-A

Table des matières

Introduction

Ce laboratoire propose une introduction aux techniques de programmation réparties asynchrones. Beaucoup plus complexes à maîtriser que les techniques synchrones, la programmation asynchrone est connue surtout dans le monde des interfaces utilisateurs.

Dans ce laboratoire nous allons illustrer l'utilisation de différentes techniques de protocoles asynchrones pour du mobile.

Pour ce laboratoire, un serveur applicatif accessible sur http://sym.iict.ch/ à été mis à disposition. Plusieurs services sont définis sur ce serveur, entre autre le plus simple (http://sym.iict.ch/rest/txt) qui implémente un service écho, on lui POSTez du texte (Content-Type: text/plain) et il nous l'envoie en retour accompagné de certaines informations sur le serveur.

Manipulation

Pour réaliser cette manipulation, il faudra au minimum une petite application avec une première activité proposant 5 boutons permettant de lancer 5 activités implémentant les 5 points suivants.

La manipulation proposée ici implique une communication asynchrone avec un serveur, sur la base d'une méthode synchrone comme le protocole HTTP (à implémenter) qui aurait, par exemple, pour signature :

public String sendRequest(String url, String request) throws Exception

Il existe aussi des librairies permettant de simplifier le développement sur Android de la communication HTTP, comme par exemple Volley. Mais dans le cadre de ce laboratoire, nous souhaitons mettre en avant l'asynchronisme et les difficultés associées, il nous est demandé de ne pas utiliser de librairie « clé-en-main ». Celles-ci permettent souvent de faciliter la mise en œuvre des cas « simples », mais dans les cas plus complexes ou sortant de l'ordinaire (par exemple la compression des requêtes sortantes) elles peuvent compliquer le problème.

Activité Asynchrone

Le but de cette activité est de pouvoir faire des requêtes asynchrones donc de continuer son exécution sans devoir attendre la réponse à une requête. Une fois la réponse reçue, une handler serait appelé avec la réponse comme paramètre.

Note : Il peut être intéressant de se tourner vers la classe AsyncTask pour résoudre l'asynchronisme. Elle a l'avantage de simplifier la communication entre les différents threads. Cependant, les AsyncTask sont exécutées individuellement les unes après les autres, cela signifie qu'un login peut se trouver bloqué en attendant que l'on charge plusieurs images lourdes.

Activité Différée

Pour cette activité, il a fallu réaliser des requêtes différées. En l'absence de connexion avec le serveur, l'application fonctionne normalement, sans que l'utilisateur n'éprouve une gêne quelconque. Dès que la connexion avec le serveur est rétablie, les informations qui avaient été fournies par l'utilisateur sont transmises au serveur.

Pour faire ceci, lors d'une demande d'envoi d'une requête nous créons un thread si ce n'est pas déjà fait. Celui-ci checkera toutes les 5 secondes s'il y a une connexion à internet. Quand la connexion sera bien présente, il enverra toutes les requêtes précédemment envoyées. Donc quand il y aucune demande de requête, le thread ne prend aucune ressource CPU.

Afin de tester cette activité, nous avons coupé notre wifi et envoyé plusieurs requêtes. Ensuite, nous avons réactivé le wifi pour voir toutes les réponses des requêtes dans les LOGs.

Cette méthode pourrait être problématique si le réseau est stable et qu'on modifie plusieurs fois le même élément car l'ordre d'arrivée des requêtes n'est pas garanti.

On pourrait utiliser le multiplexage de toutes les connexions vers un même serveur en une seule connexion. Plus d'information sur le principe ici : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-115/Multiplexage-des-connexions-SSH.

Activité Sérialisation

Le but de cette activité est de transmettre des objets. Nous allons utiliser à nouveau notre service de transmission pour envoyer un objet sérialisé sous forme de texte (xml ou json) vers le serveur, et récupérer l'information qu'il nous renvoie (serveur miroir, contenu envoyé accompagné de quelques informations sur le serveur) pour la restituer sous forme d'instance d'objet.

Pour les formats JSON le serveur http://sym.iict.ch/rest/json est utilisé.

Pour les formats XML le serveur http://sym.iict.ch/rest/xml est utilisé. Le serveur s'attend à recevoir une liste de personnes avec leurs numéros de téléphones (annuaire), les documents xml échangés seront donc vérifiés par la DTD disponible à l'adresse suivante http://sym.iict.ch/directory.dtd.

Activité Compressé

Pour cette partie, le but est d'utiliser la compression/décompression afin de réduire la quantité de donnée à envoyer et recevoir afin de réduire le temps aux requêtes de s'achever.

Afin de simuler une mauvaise vitesse d'envoie et de réception, l'en-tête X-Network est forcé à CSD(*Circuit Switched Data*).

De plus il l'a fallu ajouter l'en-tête « Content-Encoding: deflate » afin de préciser au serveur que le contenu envoyé est compressé.

Activité GraphQL

Le but de cette manipulation est de réaliser une petite interface listant les auteurs disponibles en base de données, et lorsque l'utilisateur sélectionne un auteur, d'afficher la liste de ses postes.

Pour cette activité, le serveur http://sym.iict.ch/api/graphql est utilisé avec le format JSON.

Questions

Traitement des erreurs

Les classes et interfaces SymComManager 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 classes / interfaces pour vous aider à illustrer votre réponse.

Comme dit dans la question, SymComManager et CommunicationEventListener ne gère aucun des cas d'erreurs mentionnés. En effet si le serveur n'est pas joignable ou s'il retourne un code http d'erreur.

Dans le premier cas, si le serveur n'est pas joignable l'utilisateur ne verra rien de spécial mise à part qu'aucune réponse arrive. En revanche, une exception sera levée. Afin de remédier à ça, il serait préférable de commencer par s'assurer que le téléphone soit bien connecté à internet. Ensuite, il serait possible d'utiliser le même principe que la méthode différée. Donc, d'attendre un moment en enregistrant la requête et d'essayer plus tard.

Dans le deuxième cas, si le serveur revoie un code d'erreur, il est possible de tester cela. Il y a deux cas d'erreurs possibles, les erreurs 4xx qui sont dues au serveur et 5xx qui sont dues à l'utilisateur. Dans le premier cas, l'erreur est due au serveur donc il serait possible d'appliquer la même méthode que lors d'un problème de connexion. Pour une erreur 5xx qui est donc due au client, comme nous faisons dans ce laboratoire, il est possible de récupérer l'erreur et de l'afficher à l'utilisateur afin qu'il puisse prendre conscience de l'erreur et y remédier.

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 ?

Cela peut dépendre de l'application mais en général si une authentification est nécessaire, toute activités suivant l'authentification dépendra de la réponse ce celle-ci. Donc, un protocole asynchrone ne serait pas idéal mais tout de même possible.

De plus une transmission différée peut être utilisée mais n'est clairement pas adapté à une authentification. En effet, lors d'une authentification nous souhaitons une réponse rapide et le différé n'assure pas ce principe.

L'une des restrictions d'une authentification est le temps de réponse qui se doit d'être relativement rapide afin que l'expérience utilisateur soit agréable. De plus, lors d'une authentification, des données confidentielles sont envoyées, il soit doit donc que la communication soit spécialement sécurisée.

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 ?

Contrairement aux asynctasks, les threads peuvent s'exécuter au même moment si plusieurs cœurs sont accessibles. Dans ce cas, il est important de prêter une attention particulière aux accès concurrents (sections critiques) si les threads se partagent des ressources ce qui est généralement le cas. De plus, l'ordre d'exécution des threads pourrait varier, il est donc important qu'une synchronisation se fasse afin qu'aucunes tâches s'effectuent avant qu'une dépendance ne soit pas respectée.

Ecriture différée

Lorsque l'on implémente l'écriture différée, il arrive que l'on ait soudainement plusieurs transmissions en attente qui deviennent possibles simultanément. Voici deux possibilités :

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

Pour cette dernière, il est important de bien identifier les requêtes afin de traiter correctement les réponses respectives. De ce fait, chaque réponse pourra être traitées et envoyées où elle doit se rendre de manière correcte. L'avantage de cette méthode est qu'elle permet de partager une même ressource entre plusieurs activités. De plus, on garantit une meilleure fiabilité des données. Cependant, pour des transmissions de données relativement importantes, cette méthode convient moins qu'une connexion par transmission différée.

Pour celle-ci, elle est plus utile et pratique lors d'une connexion plutôt faible et non stable. De plus, contrairement au multiplexage, cette méthode est adaptée pour l'envoie de gros volume de donnée.

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 ?*

Comme mentionné dans la question, le désavantage du type REST/JSON est qu'il n'offre pas de service de validation. Donc, cela peut donner plus de travail au développeur afin qu'il test lui-même les contraintes imposées lors de l'envoie ou la réception d'une requête/réponse.

Néanmoins, si on souhaite faire des transmissions de données sans trop de validation ou même pas du tout, elle peut être simple et rapide à implémenter.

b. *L'utilisation d'un mécanisme comme Protocol Buffers 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 ?*

Le Protocol buffers constituent le mécanisme extensible de sérialisation des données structurées de Google. Elle est bien compatible avec une architecture basée sur HTTP.

Elle est plus légère et permet une validation de la structure de donnée. De plus, elle est retro compatible et fonctionne avec d'autres produits ou systèmes existants ou futurs et ce sans restriction d'accès ou de mise en œuvre. En revanche, elle n'est pas facilement lisible pour un humain comparé au XML/JSON et est indécodable sans la structure source.

Plus d'information ici : https://blog.octo.com/protocol-buffers-benchmark-et-utilisation-sur-mobile/

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 ?*

Afin d'optimiser des ressources pour une utilisation mobile, il serait judicieux d'enregistrer des requêtes ou des réponses afin d'éviter l'utilisation excessive de la connexion avec des grandes

réponses. Il serait aussi envisageable d'utiliser la compression afin de faire transiter moins de volume de donnée a travers la connexion ce qui peut consommer beaucoup de batterie au mobile.

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 ?

Taille des données brute	**Gains pour du texte.** *Ex : coucou les amis*	**Gains pour des bytes répétitifs** *Ex : 111111aaaaasssssss8888*
10B	0.83	2
500B	~3	10
1KB	~4	20

On peut constater que plus le texte et grand plus le gain est élevé mais aussi que les données très répétitives seront beaucoup mieux compressées que du texte aléatoire.

On remarque aussi que pour les très petits messages le gain est inférieur à 1, il est donc inutile de compresser les messages de mois de 30 – 50 caractères.

Conclusion

Difficultés rencontrées

Nous avons trouvé particulièrement difficile l'introduction du laboratoire dû à la grande donnée. De plus, pour l'activité de la compression nous aurions préféré d'utiliser une méthode de Stream afin d'appliquer la compression et décompression de manière « instantané » et non pas avec un buffer avec le quel nous sommes limité pour la taille de celui-ci.

Compétences acquises

Il est vrai qu'il n'est pas simple de se lancer dans un grand laboratoire comme celui-ci mais il nous apprend à s'organiser et travailler pas à pas afin d'arriver à un but. De plus, nous avons amélioré nos connaissances autant du côté layout, android et java. Nous avons pu prendre conscience des difficultés de la programmation asynchrones.

Résultats obtenus

Finalement, nous avons une solution non pas optimisée mais fonctionnelle à 100%. Nous sommes fières de ce travail opérationnel et très instructif.

Date : 14.11.19

Nom de l'étudiant : Spinelli Isaia et Simonet Yoann