Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Georg Hinkel

Verteilte Systeme

SS 2024
LV 4132

Übungsblatt 5
Praktische Übungen
Bearbeitungszeit: 2 Wochen
Abgabe: 02.06.2024, 23:59 Uhr MESZ

Aufgabe 5.1 (Projekt „Hamsterasyl mit gRPC“, 20 Punkte):

Das selbst gestrickte RPC-Protokoll der Open-Source Bibliothek Hamsterlib, ist zwar robust und effizient, jedoch ist die Implementierung komplex und aufwändig. Das IT-Unternehmen, welches die aktuelle Implementierung des RPC-Protokolls vertreibt, verlangt horrende Preise für die Wartung. Um auf lange Sicht Geld zu sparen, beschließt das westhessische Hamsterverwahrungsunternehmen in eine standardisierte RPC-Lösung zu investieren und somit nicht mehr von einem Unternehmen abhängig zu sein. Das gRPC Protokoll ist allgemein verfügbar, es ist für gute Effizienz bekannt und es erlaubt durch die Verfügbarkeit von Implementierungen in verschiedenen Programmiersprachen und -plattformen problemlos die Weiterverwendung der Hamsterbibliothek. Deshalb entscheidet man sich für diesen Standard.

gRPC

In Ihrem Git-Repository finden Sie die bereits bekannte Hamsterlib. Unter `src` finden Sie auch den Quellcode eines einfachen Menüprogramms, das die API der Hamsterlib verwendet. Zudem ist auch eine Vorlage für den Client enthalten, in dem das Parsen der CLI-Schnittstelle bereits implementiert ist, die vom Parser aufgerufenen Funktionen aber noch leer sind.

Ihre Aufgabe ist es nun, dieses Programm in ein verteiltes Programm, bestehend aus einem **Server** und einem **Client**, umzuwandeln. Dabei sollen Server und Client mittels gRPC miteinander kommunizieren. Wichtig ist hierbei, dass die Ausgabe des neuen Clients identisch mit der Ausgabe des bestehenden Client-Programms ist.

Dazu sind folgende Schritte erforderlich:

(a) Erstellen einer RPC **Schnittstellenspezifikation** `hamster.proto` im Unterverzeichnis `proto`. Hierfür ist bereits eine Vorlage angelegt, die Sie nur noch ausfüllen müssen.

(b) Orientieren Sie sich beim Ausfüllen der Vorlage vor allem daran, was für Informationen Sie im Client benötigen. Sehen Sie keine Returncodes vor, arbeiten Sie für die Übermittlung von Fehlern stattdessen mit dem Status Aborted und nutzen Sie die Möglichkeiten in gRPC, unabhängig von der Proto-Spezifikation Fehlernachrichten zu übermitteln.

(c) Automatisches Generieren der **RPC stubs**. Details hierzu finden Sie in den technologiespezifischen Hinweisen.

(d) Manuelles Anpassen der client- und serverseitigen Templates so, dass der Server die Schnittstelle der Hamsterlib aufruft und der Client eine zu der Konsolenanwendung des ersten Übungsblatts identische, auf RPC aufgesetzte Schnittstelle bietet.

(e) Kompilieren der client- und serverseitigen Programmteile

gRPC Tests

Sie können natürlich Ihren Client mit Ihrem Server jederzeit in zwei verschiedenen Debugger-Instanzen gegeneinander manuell testen.

Zum vollständigen Testen finden Sie im Verzeichnis `tests` wieder eine in Java entwickelte Testsuite. Der Quellcode dieser Testsuite befindet sich im Ordner `Hamster_RPCClient`. Anders als der Name vielleicht vermuten lässt, wird diese Testsuite sowohl Ihren Server als auch Ihren Client mit verschiedenen Optionen aus und wertet die Ausgaben, insbesondere des Clients aus.

Wichtig: Um die Tests einfacher zu machen, wurden kleine Anpassungen an der Ausgabe des Clients vorgenommen. So wird erwartet, dass Sie bei den Verben *add*, *feed* und *bill* nur das Ergebnis (ID, übrige Leckerli oder zu zahlenden Betrag) als Ganzzahl auf die Konsole ausgeben.

Aufgrund dieser Testsuite wird dann auch wieder die Bewertung Ihrer Abgabe vorgenommen. Es ist daher wichtig, dass Sie sich genau an die vorgesehene Ausgabe halten.

Sie können für Testzwecke aber wieder ggf. die Pfade anpassen und die Tests manuell in der Entwicklungsumgebung Ihrer Wahl (vorzugsweise IntelliJ) ausführen. Das ermöglicht es Ihnen, einzelne Kommandos zu testen und die Tests zu debuggen. Ggf. empfiehlt es sich, das starten des Servers auszukommentieren oder einen Breakpoint zu setzen, um sich dann mit der Entwicklungsumgebung Ihres eigentlichen Projekts an den Server-Prozess anzuhängen, um diesen sinnvoll debuggen zu können.

Tipps zur Bearbeitung

Machen Sie sich zunächst anhand eines einfachen Beispiels mit der Benutzung von gRPC vertraut. Es gibt unzählige gRPC Tutorials im Netz. Mit [1] sei hier nur eines davon genannt. Erproben Sie das dort gezeigte Beispiel, um die Benutzung der Tools kennenzulernen.

Die Vorlagen enthalten bereits eine Proto-Datei und den notwendigen Code, um den Server und Client zu starten. Die Vorlage für den Client stellt auch bereits die richtige Verbindung zum Server her, macht aber noch nichts damit. Leider unterstützt der HTTP-Proxy der Hochschule kein HTTP/2.0, daher wurde bei der Erstellung des Kanals zum Server jeweils der Proxy explizit entfernt, da gRPC die Proxyeinstellungen sonst automatisch erfasst und die Tests auf GitLab sonst fehlschlagen.

Die Serverklasse ist ebenfalls schon angelegt aber leer. Die Methoden, die Sie überschreiben müssen, ermitteln sich aus den RPCs, die Sie in der Proto-Datei spezifizieren, es empfiehlt sich also auch hier, mit der Proto-Datei anzufangen.

Hinweise Java

In Gradle sind nun zwei Build targets hinterlegt, einmal das Ziel `hamster_server` und einmal das Ziel `hamster_client`. Beachten Sie bitte, dass das in den Vorlagen enthaltene Skript `gradle-build.sh` immer nur ein Ziel bauen kann, falls Sie also auf einem Poolrechner arbeiten,

sollten Sie immer nur ein Target auf einmal bauen oder direkt mit `gradlew` arbeiten und den Proxy direkt spezifizieren (umständlich).

In Gradle sind Plugins hinterlegt, die automatisch aus Ihren Proto-Dateien Code generieren und diesen den gängigen Entwicklungsumgebungen zugänglich machen. Das können allerdings nicht alle Entwicklungsumgebungen, daher auch hier der Hinweis auf IntelliJ. Es empfiehlt sich also, erst die Proto-Datei zu bearbeiten und dann zu kompilieren, damit der Quellcode aus der Proto-Datei generiert wird und Ihnen dann zur Verfügung steht.

Hinweise C

Für die Entwicklung mit C verwenden Sie am besten die C++-Implementierung von gRPC. Dazu müssen Sie leider zunächst das Repository klonen und gRPC lokal bauen. Dokumentation hierzu finden Sie auf der Website von gRPC: `https://grpc.io/docs/languages/cpp/quickstart/`.

Sie werden bemerken, dass in der Vorlage dieses Mal kein Makefile enthalten ist. Das liegt daran, dass in der modernen C++-Entwicklung bei der Verwendung von komplexeren Bibliotheken mit Tools gearbeitet wird, die die Makefiles generieren, bspw. CMake. Um gRPC kompilieren zu können, brauchen Sie sowieso CMake[1], also wird das auch in der Vorlage verwendet.

Wichtig: Es empfiehlt sich, gRPC wie in der Installationsanleitung beschrieben nicht global zu installieren, da Sie es sonst nur sehr schwer wieder los bekommen (es sei denn, Sie arbeiten in einem Container). Wenn Sie in einem Docker-Container arbeiten wollen, können Sie den Container nehmen, den die CI-Pipeline verwendet, da ist gRPC schon vorinstalliert. Die verlinkte Installationsanleitung schlägt bspw. `$HOME/.grpc` als Installationspfad vor. Sie müssen aber CMake beim Übersetzen Ihrer Lösung mitteilen, für welchen Installationspfad Sie sich entschieden haben. Das können Sie entweder durch den Kommandozeilenparameter oder durch Umgebungsvariablen bewerkstelligen. Die Aufrufe könnten dann also entsprechend wie folgt aussehen:

```
cmake -DCMAKE_PREFIX_PATH=$HOME/.grpc .
```

oder

```
export CMAKE_PREFIX_PATH=$HOME/.grpc
cmake .
```

Das Vorgehen mit einer Umgebungsvariable hat zudem den Vorteil, dass diese auch von anderen Programmen wie Entwicklungsumgebungen verstanden wird. Damit können Sie bspw. von CLion automatische Code-Vervollständigung bekommen, was die Bearbeitung des Übungsblattes wesentlich erleichtert.

Das CMake-generierte Makefile wird auch automatisch den Quellcode für Ihre Proto-Dateien generieren, Sie müssen also nichts weiter tun, als make aufzurufen. Das von CMake generierte Makefile enthält außerdem auch Targets für diesen generierten Code, sie können also auch jederzeit nur die RPC-Stubs generieren lassen, indem Sie make mit dem passenden Target aufrufen.

Hinweise C#

Für .NET existieren im Augenblick zwei verschiedene Implementierungen von gRPC, einmal eine verwaltete Implementierung auf Basis von ASP.NET Core und eine nicht-verwaltete Im-

[1] Die Dokumentation von gRPC kann sich nicht so recht zwischen CMake und Bazel entscheiden, auf unterschiedlichen Seiten bekommen Sie unterschiedliche Empfehlungen. Wir würden Ihnen aber CMake empfehlen, da dieses Tool nach unserer Wahrnehmung verbreiterter ist und außerdem auch direkt von Entwicklungsumgebungen wie CLion erkannt wird.

plementierung auf Basis eines in C implementierten Kernfunktionalität. Das Problem hierbei: Erstere funktioniert im Moment nicht unter MacOS, zweitere wird nicht aktiv weiterentwickelt. Wenn Sie MacOS verwenden und die Aufgabe in C# erledigen wollen, würden wir Ihnen dringend dazu raten, diese Aufgabe in einer virtuellen Maschine entweder auf Basis von Linux oder Windows zu erledigen.

Ansonsten können Sie das Tooling für gRPC in .NET bequem über NuGet-Pakete installieren. Verwenden Sie hierbei für den Server das Paket `Grpc.AspNetCore` und für den Client die Pakete `Grpc.Tools, Grpc.Net.Client` und `Google.ProtoBuf`. Damit können Sie auf die Proto-Dateien in der Projektdatei verweisen, bspw. mit

```
<Protobuf Include="..\Proto\hamster.proto" GrpcServices="Server"
          Link="Protos\hamster.proto" />
```

Die Vorlage enthält bereits eine Projektmappe mit Projekten, die diese Pakete bereits enthalten.

Die Integration von gRPC in .NET geht soweit, dass mit diesen Paketen (insb. Grpc.Tools) automatisch beim Speichern einer Proto-Datei Code erzeugt wird. Das bedeutet auch, dass wenn Sie die Datei mit einem Syntaxfehler speichern, der bisherige Code gelöscht wird und Sie jede Menge Compilerfehler bekommen.

Das Vorgehen mit verlinkten Proto-Dateien hat hier den Vorzug, dass Sie keine gemeinsam verwendeten Bibliotheken zwischen Server und Client haben müssen, was insbesondere sinnvoll ist, wenn Server und Client andere Zielarchitekturen verwenden. Leider ist die Interaktion mit der Proto-Generierung nicht so gut, Visual Studio generiert im Hintergrund den Code nur für das Projekt, in dem Sie die Proto-Datei gerade geöffnet haben.

[1] `https://grpc.io/docs/languages/java/basics/`