Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator nr.1

Programarea aplicațiilor distribuite

Tema: Agent de mesagerie

Au efectuat: st. gr. TI – 191 Gusac Dmitri

Cheptea Cătălin

Guțu Vlad

Patranjel Emanuela

A verificat: asistent universitar, Bîtca Ernest

Chişinău 2022

**Scopul proiectului:**

* studiul protocoalelor de transport TCP/IP în contextul dezvoltării unei aplicații distribuite
* studiul modelelor pentru procesarea datelor XML (DOM/ SAX) și JSON la distribuirea acestora;
* integrarea bazată pe agenți de mesaje care ar permite o comunicare asincronă dintre componentele distribuite ale unui sistem.

**Obiective:**

* definirea protocolului de lucru al agentului de mesaje;
* elaborarea nivelului abstract de comunicare (rețea) necesară elementelor pentru primirea/transmiterea mesajelor de către emițător-agent-receptor;
* elaborarea elementelor ce asigură păstrarea mesajelor primite;
* elaborarea nivelului abstract de rutare a mesajelor;
* aplicarea protocolului TCP/UDP în transmisiuni.  Aplicarea tehnicilor pentru transmisiuni unicast și multicast;
* dezvoltarea modulului de validare a datelor (în format XML).

**Ce este message broker?**

Un broker de mesaje este un modul software intermediar pentru calculator care traduce un mesaj din protocolul formal de mesagerie al expeditorului în protocolul formal de mesagerie al destinatarului. Brokerii de mesaje sunt elemente ale rețelelor de telecomunicații sau computere în care aplicațiile software interacționează prin schimbul de mesaje definite formal.

Șablonul Publisher / Subscriber

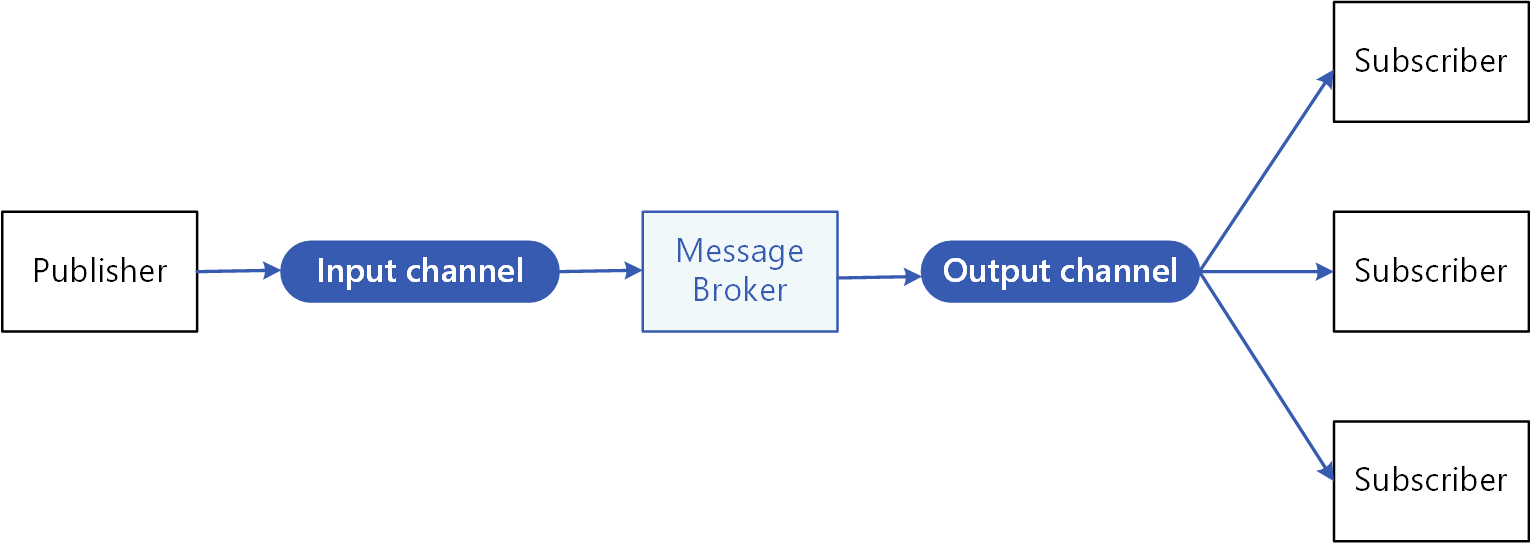
Din Figura 1 unde avem reprezentat șablonul Publisher / Subscriber observam că avem Publisher care transmite mesajele, avem 3 subscriberi care la momentul de transmitere a mesajului de abonare transmite și topicul la care face abonare, brokerul menține această conecțiune la el în memorie și când vine mesaj pe un topic anumit, brokerul redirecționează la conecțiunea existentă sau înregistrată.

Figura 1. Șablonul Publisher / Subscriber

**Mersul lucrării:**

**Partea 1**

Pentru a realiza obiectivele lucrării și a atinge scopul final avem nevoie sender, broker și receiver. Sender transmite mesaj la broker, acesta la rândul său în dependență de topic trimite mesajul la receiver. Fiecare receiver are topicul său. După cum observăm în Figura 2, brokerul este mai complex ca senderul și receiverul, el se ocupă cu rutarea mesajelor, are un storage care poate fi transient- cu ciclu de viață mai scurt sau persistent- storage care ar salva mesajele chiar dacă brokerul cade sau când primim excepții netratate.

Diagram

Description automatically generated

Figura 2. Principiul de lucru a aplicației

Protocolul de ales la realizarea proiectului este TCP (Transmission Control Protocol), pe lângă asta SocketType ales va fi Stream care se folosește pentru conecțiuni TCP.

În publisher mai avem nevoie de o metodă care ne va permite o conecțiune la broker care o să primească ca parametru adresa ip și portul deși brokerul nostru va fi local. Conecțiunea este efectuată asincron prin metoda BeginConnect care în background ar porni un alt fir de execuție care ar începe conecțiunea și când conecțiunea este efectuată, atunci se efectuează o funcție sau Callback.

Formatul mesajelor transmise va fi JSON, acest lucru va fi implementat în urma includerii unui package numit Newtonsoft.Json care ne va ajuta să serializăm și deserializăm mesajele noastre (Figura 3).

Când creăm receiverul nostru avem nevoie de citit topicul și în baza lui va fi necesar de făcut subscribe, apoi avem nevoie să facem subscribe la broker. După ce ne-am conectat cu succes, avem nevoie de o metodă care o sa facă receive, deoarece cel mai important rol al unui subscriber este să facă receive.

**Partea 2**

gRPC este un cadru RPC (Remote Procedure Call) robust open-source, folosit pentru a construi API-uri scalabile și rapide. Permite aplicațiilor client și server să comunice transparent și să dezvolte sisteme conectate. Multe firme tehnologice de vârf au adoptat gRPC.Acest cadru se bazează pe HTTP/2, buffer-uri de protocol și alte stive de tehnologie moderne pentru a asigura securitatea, performanța și scalabilitatea maximă API.

Protocol buffers, sau Protobuf, este protocolul de serializare/deserializare Google care permite definirea ușoară a serviciilor și generarea automată a bibliotecilor client. gRPC folosește acest protocol ca limbaj de definire a interfeței (IDL) și set de instrumente de serializare. Versiunea sa actuală este proto3, care are cele mai recente funcții și este mai ușor de utilizat.

Serviciile gRPC și mesajele dintre clienți și servere sunt definite în fișiere proto. Compilatorul Protobuf, protoc, generează cod client și server care încarcă fișierul .proto în memorie în timpul rulării și utilizează schema din memorie pentru a serializa/deserializa mesajul binar. După generarea codului, fiecare mesaj este schimbat între client și serviciul de la distanță.

Întregul flux arată că Protobuf oferă câteva beneficii grozave față de JSON și XML. Analiza cu Protobuf necesită mai puține resurse CPU, deoarece datele sunt convertite într-un format binar, iar mesajele codificate au dimensiuni mai ușoare. Deci, mesajele sunt schimbate mai rapid, chiar și în mașinile cu un procesor mai lent, cum ar fi dispozitivele mobile.

Diagram

Description automatically generated

Figura 5. Arhitectura gRPC

În următoarea diagramă de arhitectură gRPC, avem părțile client și server gRPC. În gRPC, fiecare serviciu client include un stub (fișiere generate automat), similar cu o interfață care conține procedurile curente de la distanță. Clientul gRPC efectuează apelul de procedură locală la stub cu parametrii care urmează să fie trimiși la server. Stub-ul clientului serializează apoi parametrii cu procesul de marshaling folosind Protobuf și trimite cererea către biblioteca locală de timp client din mașina locală.

Sistemul de operare efectuează un apel către serverul de la distanță prin protocolul HTTP/2. Sistemul de operare al serverului primește pachetele și apelează procedura server stub, care decodifică parametrii primiți și execută invocarea procedurii respective folosind Protobuf. Stub-ul serverului trimite înapoi răspunsul codificat la stratul de transport al clientului. Stub-ul clientului primește înapoi mesajul rezultat și despachetează parametrii returnați, iar execuția revine apelantului.

**Avantaje și dezavantaje ale gRPC:**

Unele avantaje și dezavantaje ale gRPC sunt relevate în *Tabelul 1*:

|  |  |
| --- | --- |
| **Avantaje** | **Dezavantaje** |
| **Performanţă:**  Prin diferite evaluări, gRPC oferă performanță și securitate API de până la 10 ori mai rapide decât comunicarea REST+JSON, deoarece utilizează Protobuf și HTTP/2. Protobuf serializează rapid mesajele de pe server și client, rezultând încărcături utile de mesaje mici și compacte. HTTP/2 mărește clasarea performanței prin server push, multiplexare și compresie antet. Server push permite HTTP/2 să împingă conținut de la server la client înainte de a fi solicitat, în timp ce multiplexarea elimină blocarea head-of-line. HTTP/2 folosește o metodă de compresie mai avansată pentru a face mesajele mai mici, rezultând o încărcare mai rapidă. | **Suport limitat pentru browser**  Deoarece gRPC utilizează în mare măsură HTTP/2, este imposibil să apelați direct un serviciu gRPC dintr-un browser web. Niciun browser modern nu oferă controlul necesar asupra solicitărilor web pentru a accepta un client gRPC. Prin urmare, un strat proxy și gRPC-web sunt necesare pentru a efectua conversii între HTTP/1.1 și HTTP/2. |
| **Streaming:**  gRPC acceptă semantica de streaming pe partea de client sau server, care este deja încorporată în definiția serviciului. Acest lucru face mult mai simplu să construiți servicii de streaming sau clienți. Un serviciu gRPC acceptă diferite combinații de streaming prin HTTP/2:   * unar (fără streaming); * streaming de la client la server; * streaming de la server la client; * streaming bidirecțional. | **Format care nu poate fi citit de om:**  Protobuf comprimă mesajele gRPC într-un format care nu poate fi citit de om. Acest compilator are nevoie de descrierea interfeței mesajului din fișier pentru a se deserializa corect. Deci, dezvoltatorii au nevoie de instrumente suplimentare, cum ar fi instrumentul de linie de comandă gRPC, pentru a analiza încărcăturile utile Protobuf pe fir, pentru a scrie cereri manuale și pentru a efectua depanare. |
| **Generarea codului:**  Caracteristica principală a metodologiei gRPC este generarea de cod nativ pentru aplicațiile client/server. Cadrele gRPC folosesc compilatorul protoc pentru a genera cod din fișierul .proto. Generarea codului este utilizată în comanda formatului Protobuf pentru definirea atât a formatelor de mesaje, cât și a punctelor finale de serviciu. Poate produce schelete la nivel de server și stub-uri de rețea la nivelul clientului, ceea ce economisește timp semnificativ de dezvoltare în aplicații cu diverse servicii. | **Fără memorie cache Edge:**  În timp ce HTTP acceptă mediatori pentru edge caching, apelurile gRPC folosesc metoda POST, care reprezintă o amenințare la adresa securității API. Răspunsurile nu pot fi stocate în cache prin intermediari. Mai mult, specificația gRPC nu face nicio prevedere și chiar indică dorința de semantică a cache-ului între server și client. |
| **Interoperabilitate:**  Instrumentele și bibliotecile gRPC sunt concepute pentru a funcționa cu mai multe platforme și limbaje de programare, inclusiv Java, JavaScript, Ruby, Python, Go, Dart, Objective-C, C# și multe altele. Datorită formatului de fir binar Protobuf și a generării eficiente de cod pentru aproape toate platformele, programatorii pot dezvolta aplicații performante, folosind în continuare suport complet multiplatform. | **Curba de învățare mai abruptă**  Multe echipe consideră că gRPC este dificil să învețe, să se familiarizeze cu Protobuf și să caute instrumente pentru a face față fricțiunii HTTP/2. Este un motiv comun pentru care utilizatorii preferă să se bazeze pe REST cât mai mult timp posibil. |
| **Securitate:**  Utilizarea HTTP/2 peste conexiunea de criptare TLS end-to-end în gRPC asigură securitatea API. gRPC încurajează utilizarea SSL/TLS pentru autentificarea și criptarea datelor schimbate între client și server. |  |
| **Fezabilitate și productivitate:**  Deoarece gRPC este o soluție RPC all-in-one, funcționează fără probleme în diferite limbi și platforme. În plus, are unelte excelente, cu o mare parte din codul standard generat automat. Acest lucru economisește timp considerabil și le permite dezvoltatorilor să se concentreze mai mult pe logica de afaceri. |  |
| **Caracteristici încorporate ale mărfurilor:**  gRPC oferă suport încorporat pentru caracteristici de mărfuri, cum ar fi schimbul de metadate, criptarea, autentificarea, termenul limită/timeout-uri și anulări, interceptoare, echilibrarea încărcăturii, descoperirea serviciilor și multe altele. |  |

*Tabelul 1*. Avantaje și dezavantaje ale gRPC

**CONCLUZII**

În urma efectuării lucrării în echipă am atins scopul propus de a elabora agenți de mesagerie care ar permite o comunicare asincronă dintre componentele distribuite ale unui sistem. De asemenea, am aprofundat cunoștințele în mecanismul de invocare la distanță. Astfel, în cadrul laboratorului, cu ajutorul mediului de dezvoltare Visual Studio și a framework-ului .NET 6 am realizat importanța protocoalelor de rețea și a gRPC în realizarea comunicării asincrone între componentele distribuite ale unui sistem.

Astfel, ne-am îndeplinit scopul de a studia protocoalele de transport TCP/IP în contextul dezvoltării unei aplicații distribuite. Am studiat modelele pentru procesarea datelor XML și JSON la distribuirea acestora. Totodată, am implementat agenți de mesaje ce ar permite o comunicare asincronă dintre componentele distribuite ale unui sistem.

Prezenta lucrare de laborator ni s-a părut interesantă și ne-a sporit vădit interesul față de programarea aplicațiilor distribuite.