Autores

- Enrique Emanuel Rezende Tavares da Silva | 11796090
- Gustavo Ryan Nasicmento Silva | 11796785
- Lucas Vilela Aleixo | 11796249
- Raul Mello Silva | 11870986

O Tema: EP de desenvolvimento de um sistema de mensageria

Optamos por um EP de desenvolvimento, escolhemos um sistema de mensageria como tema, inspirado no Apache Kafka. Mensageria é um conceito que define que sistemas distribuídos possam se comunicar por meio de troca de mensagens (evento), sendo essas mensagens gerenciadas por uma aplicação.
Nosso sistema de mensageria se divide em: Producer, a aplicação que produz as mensagens, o Broker, que é responsável por gerenciar as mensagens, garantindo que cada mensagem seja enfileirada e armazenada até que seja consumida e o Consumer, que consome as mensagens de um tópico específico.
Supondo uma situação real como exemplo em que duas aplicações A e B precisam se comunicar, para realizar a comunicação entre essas duas aplicações sem acoplamento direto entre elas, utiliza-se um Sistema de Mensageria, dessa forma a aplicação A atua como Producer, enviando uma mensagem (evento), o Sistema de Mensageria guarda e enfileira a mensagem para que a aplicação B possa consumi-la. Da mesma maneira, pode haver um produtor de mensagens na aplicação B e um consumidor na aplicação A.

Protocolos de comunicação

Em nossa aplicação, os brokers simultaneamente esperam mensagens de um produtor, e enviam mensagens para um consumidor.

Produtor

Ao rodar um produtor, o usuário tem duas opções de comportamento: Produzir uma mensagem em um tópico ou encerrar a conexão.
Caso opte por encerrar a conexão, basta enviar:

```
exit()
```

Caso opte por produzir uma mensagem em um tópico, deve enviar uma mensagem no seguinte modelo:

```
[topicname];[body]
```

onde:

- `[topicname]` é uma string que corresponda a um tópico novo.

- `[body]` é uma string que corresponda a nova mensagem a ser criada naquele tópico.

Por exemplo, `"olá mundo;hello world!"` é uma mensagem válida que cujo nome de tópico é "olá mundo" e cujo conteúdo é "hello world!".

Consumidor

Ao rodar um consumidor, o usuário não precisa se preocupar em digitar nenhum input: Em cada inicialização do consumidor é preciso informar o tópico que será consumido. Ainda assim, por debaixo dos panos, cada produtor envia continuamente para o broker uma requisição padronizada, que comunica para esse qual foi a última mensagem lida pelo produtor naquele tópico.
Tal mensagem segue o seguinte modelo:

```
topic=[topic];lastMessageIndex=[index]
```

onde:

- `[topic]` é uma string que diz de qual tópico o broker deve resgatar a mensagem.
- `[index]` é um long que indica qual o id da última mensagem lida.

Por exemplo, `"topic=olá mundo;lastMessageIndex=0"` é uma mensagem válida que indica ao broker que a última mensagem lida por esse produtor no tópico foi a mensagem "olá mundo", e que nenhum .

Broker

As únicas mensagens que o broker envia são mensagens para cada consumidores inscritos em cada tópico.
Caso não haja novas mensagens naquele tópico:

- `"no new message"`

Caso haja mensagens novas, para cada mensagem o broker a envia de acordo com o seguinte modelo:

```
[id];[topic];[body];[datetime]
```

onde:

- `[id]` é o número (long) de identificação dessa mensagem.
- `[topic]` é o nome do tópico da mensagem, em string.
- `[body]` é o conteúdo da mensagem, em string.
- `[datetime]` é a representação da data que a mensagem foi processada pelo Broker, em LocalDateTime.

A seguinte mensagem é válida:
`"3;topicoBemLegal;essa string é o corpo da mensagem que estava salva lá no BD do broker com id 3 e tópico topicoBemLegal, logo em seguida vem o momento em que ela foi processada;2021-11-03T14:35:12.236576"`

Como compilar

O código do Kafta foi escrito em Java, utilizando SQLite como banco de dados. Para compilá-lo, é preciso apenas estar com a ferramenta `make` instalada no computador e executar o seguinte comando no diretório raíz:

- **No Linux (ou MacOS)**

```
make compile-linux
```

- **No Windows**

```
make compile-windows
```

Vale pontuar que, como são três aplicações diferentes, cada uma possui um makefile próprio em seus diretórios. Isso permite que cada aplicação seja compilada e executada fácil e independentemente das outras, e em computadores diferentes.

Como executar

Disponibilizamos para execução arquivos `.jar` para cada aplicação dentro da Kafta, ou seja, um executável para o Broker, um para o Consumer e outro para o Producer. Cada jar deve ser executado através do comando:

```
java -jar /path/to/jarfile
```

Alternativamente, caso queira executar depois de ter compilado por si mesmo, basta executar o seguinte comando, no diretório da aplicação:

```
make run
```

ou no diretório raíz:

```
make run-[which]
```

onde [which] deve ser substituído por um dos seguintes:

- broker
- producer
- consumer

Primeiramente execute o Broker, logo após o Producer e o Consumer. Após a execução do Producer, será requisitado do usuário um input, ao passo que o mesmo deve digitar a mensagem e seu tópico, como especificado em Protocolos de Comunicação - Produtor.

Como ler

Para cada pasta, "kafta-broker", kafta-consumer" e "kafta-producer", existe uma classe de Application, que é o ponto de entrada de cada aplicação. Deve-se ler os códigos a partir delas.

Testes e Resultados

Testamos a funcionalidade de cada classe e seus métodos através de testes unitários. Além disso, executamos testes de ponta a ponta manualmente, e gravamos os resultados.

Próximos passos

Reduzir a quantidade de mensagens trocadas entre consumidor e broker. Banco de dados com persistência de tópicos, melhorar o sistema de atribuir ID às mensagens.

Bibliografia

As seguintes referências foram consultadas para o desenvolvimento do Kafta, porém, nenhum trecho de código foi copiado dos sites.

https://www.geeksforgeeks.org/socket-programming-in-java/
https://www.baeldung.com/a-guide-to-java-sockets
https://www.devmedia.com.br/como-criar-um-chat-multithread-com-socket-em-java/33639
https://docs.oracle.com/javase/tutorial/networking/sockets/readingWriting.html
https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html