Computação Distribuída

Diogo Nuno Pereira Gomes
José Nuno Panelas Nunes Lau

Message Broker

Hugo Paiva, 93195
Carolina Araújo, 93248

DETI
Universidade de Aveiro
22-04-2020

1 Introdução

Este trabalho prático foi desenvolvido com o objetivo de implementar um *Message Broker* capaz de interligar produtores (*Producers*) e consumidores (*Consumers*) através de um protocolo *PubSub* comum e de três mecanismos de serialização distintos (*XML*, *JSON* e *Pickle*). Foi também desenvolvido um *Middleware* para abstrair os produtores e consumidores de todo o processo de comunicação.

2 Protocolo

Todas as mensagens enviadas são enviadas através do protocolo *TCP* na porta 8000 e precedidas de um *header* que informa ao recetor qual o número de *bytes* que a mensagem terá. Esta metodologia é crucial para a comunicação entre as várias entidades do trabalho prático pois permite receber sempre a mensagem na sua integridade. Seria possível, também, enviar a mensagem com separadores, no entanto o grupo preferiu a abordagem do *header* devido à flexibilidade que permite ter quanto ao tamanho da mensagem.

Foi definido que todos os produtores e consumidores, ao inicializarem a respetiva *queue*, enviam uma mensagem ao *Message Broker* com o mecanismo de serialização que suportam. Esta mensagem inicial é codificada em *utf-8* com o nome do mecanismo de serialização em texto simples. Apesar de ser possível enviar a mensagem inicial num mecanismo específico, como *XML*, *JSON* ou *Pickle*, foi escolhida esta abordagem pois desta forma evita-se qualquer tipo de incompatibilidade entre a comunicação das duas entidades, para além do facto de não ser necessário qualquer tipo de operação extra para a descodificação. Ao receber esta mensagem, o *Message Broker* guarda, para cada conexão diferente, o seu respetivo mecanismo de serialização.

Após esta mensagem inicial, todos os consumidores e produtores comunicam com o *Message Broker* através do seu mecanismo de serialização específico, tal como pedido no enunciado do trabalho. Em cada mensagem posteriormente enviada, é definida uma operação (*op*), um tópico (*topic*) e um valor (*value*) para representar o que é pretendido enviar.

3 Estruturas de dados

3.1 Tree

Foi criada uma classe *Tree* com o objetivo de implementar uma árvore hierárquica para os vários tópicos hierárquicos, onde cada instância desta classe representa um nó da mesma. Estas instâncias contêm os seguintes dados:

- **Path** - Tópico completo deste nó, desde a raiz até ao tópico do nó em si;
- **Topic** - Último subtópico do *path* completo do nó. Seja o path de um dado nó */weather/temperature/degree*, o tópico a ele associado sera apenas *degree*;
- **Last_msg** - Última mensagem que foi produzida para o tópico deste nó;
- **Consumers** - Lista de consumidores subscritos ao tópico deste nó;
- **Children** - Lista de todos os nós descendentes diretos;
- **Parent** - Guarda, se existir, o nó do qual o próprio provém.

Foi escolhida esta abordagem quanto à estrutura da árvore porque, visto que um consumer pode estar subscrito a mais do que um tópico, o mais lógico seria instanciar um tipo de estrutura de dados hierárquica para guardar as diferentes entidades, facilitando a criação das funcionalidades necessárias, em vez de tentar colocar tudo numa estrutura como dicionários, listas, etc.

Com esta estrutura é-nos possível percorrer a árvore com o intuito de, por exemplo: encontrar todos os nós (tópicos) a que um consumer está subscrito; procurar, entre filhos de um nó, aquele cujo subtópico corresponde ao argumento passado, o que, por sua vez, permite percorrer a árvore entrando apenas nos nós sucessivos descritos por um *path*. Torna-se ainda viável retornar todos os tópicos que têm início no nó em questão, bem como listar todas as últimas mensagens associadas a um tópico que, tendo descendentes, serão também estas listadas, etc.

Dito isto, havendo um tópico de onde brotam vários outros, pressupõe-se que, aquando a publicação de novas mensagens nesse tópico-pai, todos os consumers subscritos aos nós-descendentes do mesmo, têm de receber essas novas publicações. Assim sendo, escolheu-se adicionar os consumers do tópico do nó-pai, apenas nesse nó, e remover qualquer subscrição nos nós-descendentes. Isto foi feito com o objetivo de diminuir os tempos de procura ao longo da árvore.

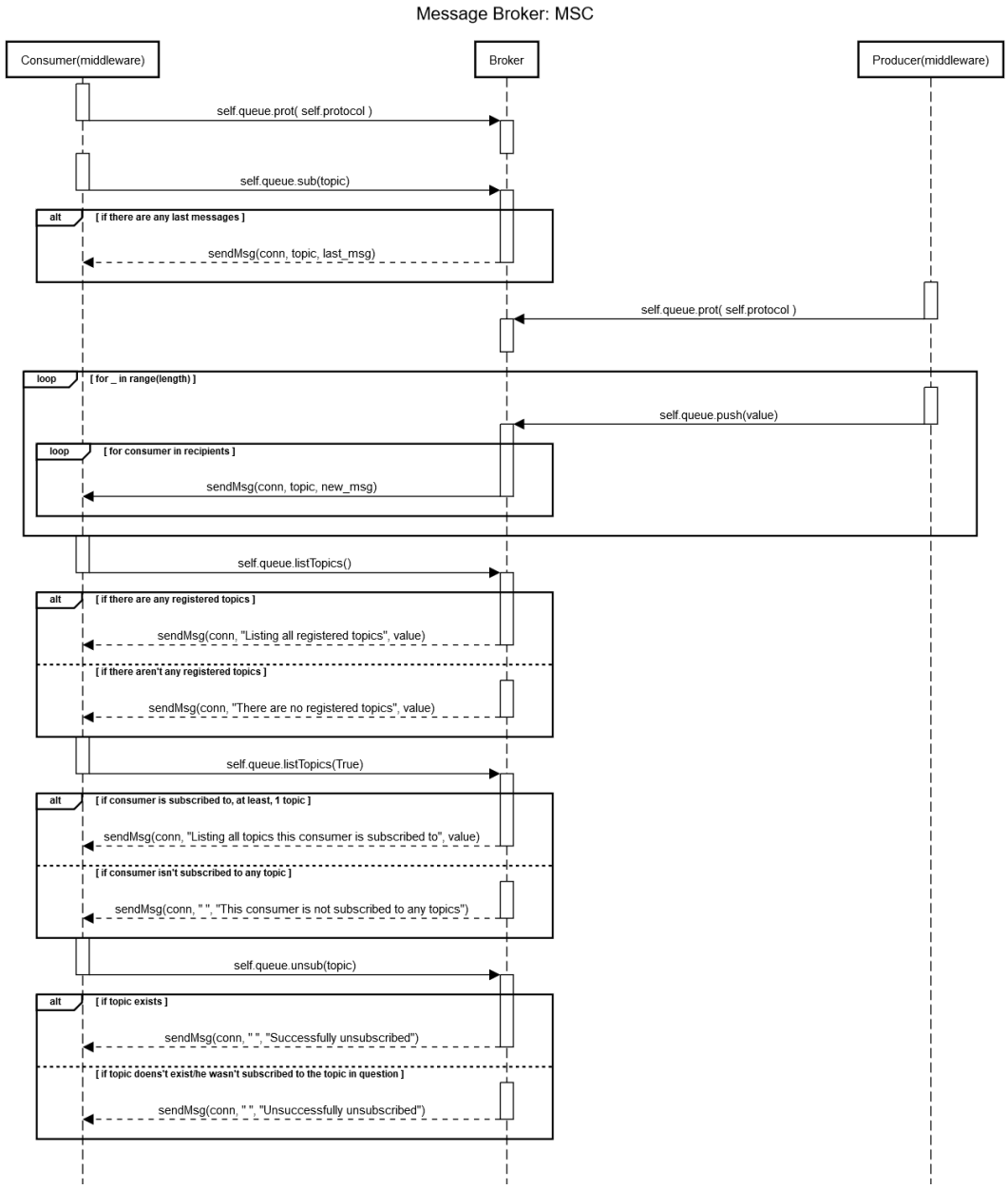
3.2 Message Broker

O Broker foi construído de modo a controlar o acesso à informação guardada na árvore, utilizando, para o efeito, uma lista que contém a *root* de cada *tree branch* dos vários tópicos, devido às vantagens mencionadas anteriormente.

Por outro lado, o *Message Broker* usa um *deque* que guarda os consumidores e produtores que esperam ser atendidos. Desta forma, evitam-se problemas de consistência de dados ao longo da executação dos vários pedidos, uma vez que os pedidos são executados na ordem de chegada e um a um. Apesar de se poder usar outro tipo de resoluções, como um mecanismo de *mutex*, foi escolhida a utilização de um *deque* por ser mais prático e simples. Para evitar erros relativamente ao envio de mensagens para consumidores que entretanto já não estão ativos, sempre que alguma conexão termina, são removidas todas as entradas no *deque* que a contém. Visto que existe uma instância diferente da classe *Queue* para cada produtor e consumidor, fez, também, mais sentido aplicar uma abordagem deste género no *Message Broker* em vez, de por exemplo, um mecanismo de *queue* na classe *Queue*.

Por fim, são guardados os mecanismos de serialização de cada conexão num dicionário devido ao fácil e rápido acesso que este tipo de estrutura de dados providencia.

4 Arquitetura de Software



Ao serem executados, tanto um consumidor como um produtor criam uma nova instância da classe *Queue* para comunicar com o *Message Broker*, tal como referido na secção do Protocolo.

A figura acima representa o fluxo principal de mensagens do trabalho prático realizado. Resumidamente, tanto os *Consumers* como os *Producers* indicam inicialmente o seu protocolo ao *Message Broker*, havendo distinção no facto de apenas os *Consumers* fazerem, logo de seguida, uma ou mais subscrições em tópicos através da função *sub(topic)*.

Para a publicação de mensagens, um *Producer* utiliza a função *push(value)*, enviando os vários valores gerados na(s) sua(s) queue(s). Acrescentaram-se ainda as mensagens de listagem de tópicos e cancelmento de subscrições, por parte dos *Consumers*, para ser possível ver, de modo simplificado, quais as particularidades destas funções.

5 Interface de listagem de tópicos e cancelamento de subscrição

Optou-se por criar duas opções de listagem de tópicos: listar todos os tópicos registados no *Message Broker* (*listTopics()*) ou listar todos os tópicos a que está subscrito o consumer que faz o pedido de listagem (*listTopics(True)*), ambas as funções definidas na classe *Queue*. Esta escolha foi feita porque se considerou que ambos seriam uma mais-valia no que toca à verificação da existência, ou não, de erros.

Quanto ao cancelamento de subscrição, novamente foram criadas duas opções, sendo que uma delas cancela a subscrição de um tópico específico (*unsub(topic)*) e a outra cancela a subscrição de todos os tópicos a que o *Consumer* estava previamente inscrito (*unsub("\")*), ambas as funções definidas na classe *Queue*. Chegou-se à conclusão de que, se era possível subscrever a todos os tópicos registados no *Broker*, então se poderia, de igual modo, cancelar o registo de todos os tópicos.

6 Estratégia de Validação da Solução

Para validar se a nossa solução correspondia ao esperado, cada vez que se atingia um milestone, os elementos do grupo corriam o *Message Broker*, seguido tanto de um ou mais *Consumers* ou *Producers*, variando, entre estes dois, aquele que era primeiro iniciado. Eram também alterados, se necessários, os códigos base destas duas entidades, testando assim funções como *sub*, *unsub*, *listTopics()*, etc. Deste modo, foi possível encontrar vários erros e/ou chegar à conclusão de que faltava algo essencial.

A par disso, o grupo criou também, na pasta *Testes*, um script de *BASH* que executa em simultâneo 5 *Consumers* e 5 *Producers* que subscrevem e publicam a tópicos diferentes, testando ainda funcionalidades como *list* e *unsub*.

7 Conclusão

Em tom de remate, de acordo com as metas especificadas pelos docentes, pensa-se que o trabalho foi bem sucedido. Uma das dificuldades mais sentidas pelo grupo foi conseguir reunir todos os casos a que a nossa solução pode ser submetida e fazer os possíveis para que esta corresponda ao esperado, o que se pensa ter sido alcançado. Foi particularmente custoso saber que muito do trabalho que foi feito acabou por ter de ser riscado da versão final, exatamente pelo motivo previamente mencionado.

Considera-se, também, que este trabalho prático ajudou a aprofundar os conhecimentos sobre o funcionamento de um *Messsage Broker*, bem como adquirir novos conhecimentos relacionados com o mesmo.